Especialización en Ciencia de Datos

TRABAJO FINAL INTEGRADOR

Clasificación de Clientes por Umbral Superior de Ingresos

Alumno: Sebastián Aguilera
Tutor: Dra. Leticia Gómez

Ciudad de Buenos Aires, Junio 2022

Índice de Contenidos

1. Introducción

El presente trabajo tiene por objetivo proveer de nuevas herramientas al análisis de datos en el proceso de toma de decisiones concerniente a la banca comercial minorista.

La propuesta está relacionada con la realización de un análisis que contenga el resultante de distintos modelos predictivos aplicados un set de datos provistos por la entidad, y una etiqueta generada por NOSIS[1] (la variable a regresar).

En general los bancos que operan como agentes financieros del sector público cuentan con diferentes fuentes de datos provistos por distintos organismos al momento de generar el alta de clientes. Este proceso se denomina “altas masivas” y, por lo común, la información provista para el alta se atiene sólo a los aspectos indispensables relacionados a la identificación de las personas.

En esta línea, desarrollar una “analítica” de la información que atienda diferentes necesidades del negocio, surge como un elemento necesario y primordial para mejorar la calidad del dato obtenido al momento del alta. Esta tarea supone el cabal conocimiento de las falencias de las bases de datos de la entidad y al mismo tiempo, una estrategia de gobierno de datos a fin de lograr la consistencia del análisis, el cual puede ser lineal o descriptivo, o como el caso que nos ocupa, de naturaleza no lineal.

En sentido estricto, la transformación digital puesta en marcha en la industria financiera local, proceso acelerado durante el escenario de pandemia, se sustenta en cuanto su desarrollo refiere a las denominadas “experiencias de usuario”. En ésta denominación convergen distintos aspectos inherentes a la forma en la que el cliente opera con la entidad y en sus caracterizaciones más necesarias a fin de poder consolidar una propuesta de valor. Este fenómeno explica la proliferación de las billeteras electrónicas y las posibilidad de realizar “on boarding” digital sin necesidad de concurrir a alguna entidad financiera.

La contracara de lo señalado en el párrafo anterior. tiene que ver con el gobierno de datos de las entidades y la conformación de equipos de trabajo que se orienten a

[1] https://www.nosis.com/es/institucional/quienes-somos

un desarrollo más prominente de la estadística descriptiva y la posibilidad de calibrar modelos de regresión no lineal, que puedan contribuir a fundamentar opciones de negocio. En general, predominan las regresiones propias de la estadística aplicada a la economía y las finanzas orientadas al planeamiento estratégico. La constitución de un repositorio único de datos constituye un desafío que empieza a ser abordado de manera más enérgica por parte de los bancos.

El presente trabajo pretende abordar alguna de las cuestiones a ser resueltas por las entidades, y que refiere a una tipología de criterio de segmentación específica sustentada en los ingresos presuntos.

2. Contexto

Los análisis de segmentación de cartera de clientes son muy variados y responden a diferentes niveles de desarrollo de los sistemas de información de las entidades.

En general, las herramientas que proveen a los mismos tienen que ver con dimensiones de base de datos y *"queries"* que regresen sobre una etiqueta determinada. Esto podría ser considerado, para nuestros fines, como una modalidad tradicional de análisis, generalmente aplicada a aspectos tácticos de la política comercial o de riesgos.

El problema que se suscita tiene que ver con el hecho de que prescinden de herramientas superiores para concentrar la acción y maximizar el rendimiento sobre el capital invertido.

3. Presentación del Problema

El análisis de datos para la toma de decisiones en lo concerniente a la banca comercial minorista es un problema recurrente. El mayor interrogante se centra en la correcta segmentación de toda la cartera de clientes minoristas del banco y la posibilidad de, a partir de algún criterio algorítmico, proceder a una mejor segmentación de dicho universo.

En tal sentido, es conveniente resaltar que el Banco utiliza el mismo criterio de segmentación que realiza NOSIS, en cuanto a criterios de ingresos. El problema radica en que la segmentación que realiza el Banco considera solamente el salario líquido ingresado en la cuenta y NOSIS utiliza todas las fuentes de información disponible (fiscales, laborales y crediticias), y además, detenta enriquecedores de fuente de datos a partir de información externa suministrada por otras fuentes (no tradicionales), como ser, supermercados o comercios. Es decir, existen clientes que en NOSIS figuran como “A” y el Banco los considera como “D1”.

4. Justificación del Estudio

El problema planteado tiene sentido en la medida que se busca determinar el nivel y segmento socioeconómico de una persona utilizando solamente información operacional de clientes, a partir de los registros transaccionales que registra la entidad.

5. Alcances del Trabajo y Limitaciones

El presente trabajo se centra en la segmentación de los clientes del Nuevo Banco del Chaco SA, a partir de algún modelo predictivo, estableciendo un criterio de clasificación binario que identifique si cada uno corresponde o no al segmento “A”.

El conjunto de datos a contemplar, se encuentra acotado a los clientes respecto de los cuales se haya formulado algún análisis crediticio o que ya hayan tomado alguna línea de crédito por canales automáticos. Es decir, clientes que cuenten con algún *“scoring”* e informe de NOSIS respectivo.

6. Objetivos

6.1. Objetivo General

Desarrollar un modelo que permita determinar el nivel y segmento socioeconómico de una persona a partir solamente de los registros transaccionales que registra la entidad.

6.2. Objetivos Específicos

- Aplicar un proceso ETL para preparar el set de datos para el análisis.
- Modelar y entrenar distintos modelos predictivos.
- Definir las métricas que permitan la comparación entre los distintos modelos utilizados.
- Determinar, acorde a las métricas propuestas, es el algoritmo que mejor se ajuste al objetivo general planteado.

7. Metodología Aplicada

A los fines de detectar el modelo que mejor se ajuste al problema a resolver, implementaremos tres algoritmos predictivos sobre el mismo *dataset.*

7.1. Modelo Conceptual

Tal como se indicó en la Sección 5, la cantidad de casos sobre el cual se realizará el modelado, se encuentra acotada a los clientes respecto de los cuales se haya formulado algún análisis crediticio (o pedidos de informe), o hayan tomado alguna línea de crédito por canales automáticos (en tal caso, se dispara automáticamente una solicitud para realizar el *scoring* y asignar el monto).

En relación a la fuente de datos generada por la entidad, la misma refiere a un *dataset* 55 campos y un promedio superior a los 37 mil registros, en el cual se encuentra contenida la identificación y segmentación de cada cliente de la banca comercial minorista, así como gran parte de su registro transaccional al cierre de cada mes. Esta fuente de datos es relativamente nueva (se creó a partir de Octubre del año 2020), y se generó ante la necesidad de evaluar el comportamiento de la cartera comercial (calibrar de mejor manera las decisiones concernientes a los diferentes productos que ofrece la Banco).

En la Figura 1 se puede consultar un fragmento del mencionado *dataset.*

CUIT	NIVEL_SOC	SCORE	# Conven	Sucursal	DNI	CP	Saldo Ant	Acred Sueldo	Otras Acred	Acr. Plazo Fijo	Pago Prestam	Pago T Credito	Compras T. Debito	Debitos Directos	Deb. Mutuales	Pagos de Serv Link	Extracc ATMs	Extracc. Caja H	...
....	C3	564	1	30	4E+06	3500	2131.72	41321.07	387901.49	0	0	0	3227.67	1131.27	0	0	0	426400	...
....	C3	664	1	30	4E+06	3500	2131.72	41321.07	387901.49	0	0	0	3227.67	1131.27	0	0	0	426400	...
....	C1	1	3	30	5E+06	3500	85989.3	113553.57	177914.03	0	0	71457.8	16740.25	13237.68	0	0	76000	0	...
....	C1	1	3	30	5E+06	3500	85989.3	113553.57	177914.03	0	0	71457.8	16740.25	13237.68	0	0	76000	0	...
....	C1	1	3	30	5E+06	3500	85989.3	113553.57	177914.03	0	0	71457.8	16740.25	13237.68	0	0	76000	0	...
....	C1	1	3	30	5E+06	3500	85989.3	113553.57	177914.03	0	0	71457.8	16740.25	13237.68	0	0	76000	0	...
....	C3	843	1	30	8E+06	3500	47364.87	217727.5	207.48	1541917.8	0	0	10893.57	8026.93	1334	0	58959.98	0	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	C2	597	1	16	1E+07	3500	135448.19	208790.34	20004.9	0	0	0	53246.3	0	1092	0	0	100000	...
....	B	341	2	30	1E+07	3500	68758.72	249383.71	594489.06	0	283449.7	258069.93	23057.95	0	704	0	26000	0	...
....	A	1	0	NULL	1E+07	3500	1330038.49	0	1688741.42	0	0	136410	0	0	0	1855395.07	0	900000	...
....	A	1	0	NULL	1E+07	3500	1330038.49	0	1688741.42	0	0	136410	0	0	0	1855395.07	0	900000	...
....	C2	577	0	NULL	1E+07	3712	4238571.87	0	2300219.13	0	0	0	0	0	0	0	0	5800000	...
....	C1	453	1	30	1E+07	3500	15436.62	349402.22	58315.18	0	17302.88	74767.82	30130.26	3207.05	0	114000	6700	0	...
....	B	716	0	NULL	1E+07	3500	336279.71	0	712178.76	0	0	27024.86	73015.48	7390.25	0	152638.29	0	300000	...
....	B	717	0	NULL	1E+07	3500	336279.71	0	712178.76	0	0	27024.86	73015.48	7390.25	0	152638.29	0	300000	...
....	B	481	1	30	1E+07	3500	37573.53	113087.66	4.92	0	0	46125.05	12006.95	800	0	0	3500	0	...
....	B	761	1	30	1E+07	3500	37573.53	113087.66	4.92	0	0	46125.05	12006.95	800	0	0	3500	0	...
....	C3	691	0	NULL	1E+07	3503	18881.83	0	4809315.7	0	0	0	86993	31853.75	0	237427.04	93530	1349000	...
....	C3	692	0	NULL	1E+07	3503	18881.83	0	4809315.7	0	0	0	86993	31853.75	0	237427.04	93530	1349000	...
....	C1	782	2	27	1E+07	3541	160771.26	196680.28	14.08	1096549.84	34193.55	32265.82	0	4498.59	0	0	0	50000	...
....	C1	747	2	27	1E+07	3541	160771.26	196680.28	14.08	1096549.84	34193.55	32265.82	0	4498.59	0	0	0	50000	...
....	B	296	1	30	1E+07	3500	33395.51	242880.16	22004.08	0	7018.93	10877	32188.65	0	429.26	37100	60000	0	...
....	B	296	1	30	1E+07	3500	33395.51	242880.16	22004.08	0	7018.93	10877	32188.65	0	429.26	37100	60000	0	...
....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....	....

Fig. 1 – Fragmento del dataset

7.2. Algoritmos a Utilizar

Los modelos que se van a utilizar en este trabajo, para ser comparados con el objetivo de detectar el mejor comportamiento, son:

- Gradient Boosting (LightGBM)
- Red Neuronal
- Support Vector Machine

7.2.1. LightGBM

LightGBM es una implementación de árboles de decisión potenciados por gradientes (GBDT), que es un método de "*ensemble*" que combina árboles de decisión (como aprendices débiles), de forma serial (potenciación) [5].

Los árboles de decisión se construyen dividiendo las observaciones (es decir, instancias de datos) en función de los valores de *"features"* o etiquetas. Así es como un árbol de decisión "aprende". El algoritmo busca la mejor división que resulte en la mayor ganancia de información.

Encontrar la mejor división resulta ser la instancia que consume más tiempo del proceso de aprendizaje de los árboles de decisión. Los dos algoritmos que se utilizan para encontrar las mejores divisiones son Preordenados y Basado en histograma. En el primero, los valores de las "*features*" se ordenan previamente y se evalúan todos los

puntos de división posibles. En el segundo, las *"features"* continuas se dividen en contenedores discretos que se utilizan para crear histogramas de características.

El algoritmo basado en histogramas es más eficiente en términos de consumo de memoria y velocidad de entrenamiento. Sin embargo, ambos algoritmos se vuelven más lentos a medida que aumenta la cantidad de instancias.

El punto de partida de LightGBM es el algoritmo basado en histogramas. Para cada "*feature*", se escanean todas las instancias de datos para encontrar la mejor división con respecto a la ganancia de información. Por lo tanto, la complejidad del algoritmo basado en histogramas está dominada por la cantidad de instancias de datos y características. Para superar este problema, LightGBM utiliza las siguientes dos técnicas: GOSS[2] y EFB[3].

GOSS (muestreo de un lado de gradiente)

Si se logra muestrear datos en función de la obtención de información, en lugar de escanear todas las instancias de datos, el algoritmo será más efectivo. GOSS aborda este problema mediante el uso de gradientes que nos brindan información valiosa sobre la ganancia de información:

- *Gradiente pequeño:* el algoritmo se ha entrenado en esta instancia y el error asociado con él es pequeño.
- *Gradiente grande:* el error asociado con esta instancia es grande, por lo que proporcionará más ganancia de información. Podemos eliminar las instancias con gradientes pequeños y solo enfocarnos en las que tienen gradientes grandes. Sin embargo, en ese caso, la distribución de datos cambiará. No queremos eso porque afectará negativamente la precisión del modelo aprendido.

GOSS proporciona una forma de muestrear datos en función de los gradientes teniendo en cuenta la distribución de datos. Las instancias de datos se ordenan según el valor absoluto de sus gradientes y se seleccionan las mejores instancias de ax100%. De las instancias restantes, se selecciona una muestra aleatoria de tamaño

[2] Gradient-based One-Side Sampling
[3] Exclusive Feature Bundling

bx100%. Dicha muestra de pequeños gradientes se multiplica por una constante igual a (1-a) / b cuando se calcula la ganancia de información [5].

Lo que GOSS finalmente logra es que el enfoque del modelo se incline hacia las instancias de datos que causan más pérdidas (es decir, con bajo entrenamiento) sin afectar mucho la distribución de datos.

EFB (paquete de características exclusivas)

Es probable que los conjuntos de datos con una gran cantidad de *features* tengan *sparse features* (por ejemplo, muchos valores cero). Estas *sparse features* suelen ser mutuamente excluyentes, lo que significa que no tienen valores distintos de cero simultáneamente. Considerar el caso de los datos de texto codificados con la técnica *one-hot encoding*. En una fila en particular, solo una columna que indica una palabra específica es distinta de cero y todas las demás filas son cero.

EFB es una técnica que utiliza un algoritmo denominada *"greedy"* para combinar (o agrupar) estos elementos mutuamente excluyentes en una sola característica (por ejemplo, un paquete de características exclusivas) y, por lo tanto, reducir la *"dimensionalidad"*. EFB reduce el tiempo de entrenamiento de GDBT sin afectar mucho la precisión porque la complejidad de crear histogramas de características ahora es proporcional a la cantidad de paquetes en lugar de la cantidad de características (la cantidad de paquetes es mucho menor que la cantidad de features) [5].

En síntesis, el objetivo es reducir el tamaño de las instancias de datos y sus features, preservando la información los máximos posibles. Las técnica GOSS y EFB son implementadas para alcanzar este objetivo.

7.2.2. Support Vector Machines

Si bien los Support Vector Machines (SVM o vectores soporte) suelen considerarse como algoritmos de clasificación, pueden emplearse tanto en problemas de clasificación como de regresió, dado que puede manejar fácilmente múltiples variables continuas y categóricas. El núcleo de SVM es encontrar un hiperplano marginal

máximo (MMH) en un espacio multidimensional que divida el conjunto de datos en clases minimizando el error. En la Figura 2 se puede ver un esquema de lo descripto.

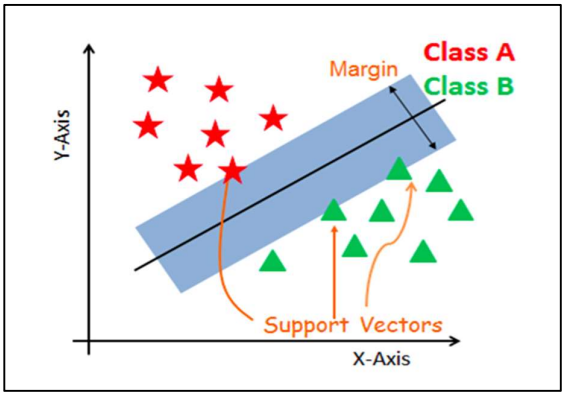

Fig. 2 - Esquema de hiperplano en SVM
Fuente:https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

Los puntos de datos que están más cerca del hiperplano se denominan vectores soporte. Estos puntos definirán mejor la línea de separación al calcular los márgenes, y serán los más relevantes para la construcción del clasificador.

El margen es un espacio entre las dos líneas en los puntos de clase más cercanos, y se calcula como la distancia perpendicular desde la línea hasta los vectores de apoyo o los puntos más cercanos. Un margen es mejor, cuanto mayor es entre las clases.

El objetivo principal del método es segregar el conjunto de datos dado de la mejor manera posible. SVM busca el hiperplano marginal máximo a través de los siguientes pasos: Genera hiperplanos que segregan las clases de la mejor manera, y luego selecciona el hiperplano correcto con la segregación máxima desde los puntos de datos más cercanos. En la Figura 3, a la izquierda, se muestran tres hiperplanos (negro, azul y naranja), donde el azul y el naranja tienen un mayor error de clasificación, pero el negro separa las dos clases correctamente; a la derecha se puede ver la selección del hiperplano correcto.

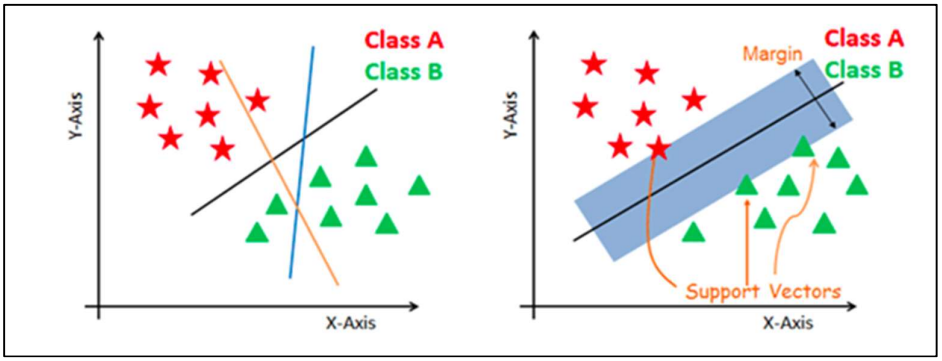

Fig. 3 - Esquema de hiperplanos en SVM

Fuente:https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

Algunos problemas no se pueden resolver usando un hiperplano lineal. En tal situación, SVM utiliza mecanismo para transformar el espacio de entrada en un espacio dimensional superior. Los puntos de datos se trazan en el eje x y el eje z (z es la suma al cuadrado de $x, y: z = x^2 + y^2$). Una vez realizado eso, se puede segregar fácilmente esos puntos mediante la separación lineal. En la Figura 4, se puede ver esquemáticamente.

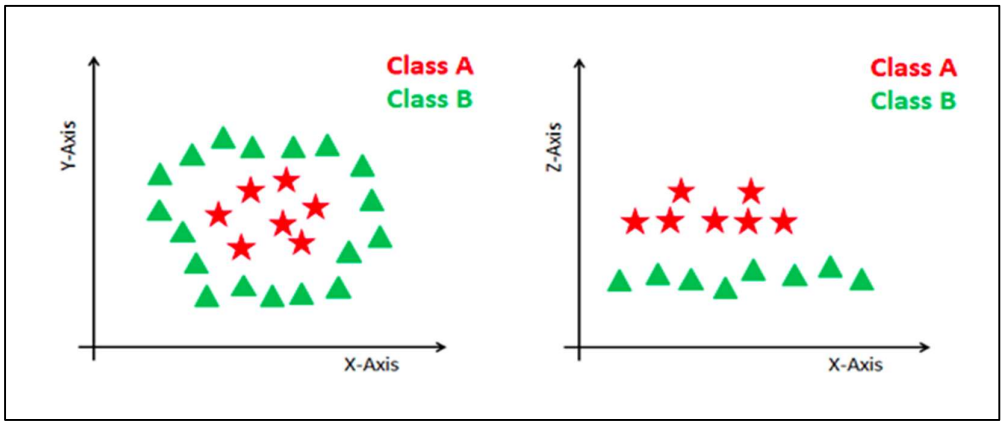

Fig. 4 – Transformación de espacio de entrada en un espacio dimensional superior

Fuente:https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python

En la práctica, el algoritmo SVM se implementa usando un *kernel*. El *kernel* toma un espacio de entrada de baja dimensión y lo transforma en un espacio de mayor dimensión, convirtiendo un problema no separable en un problema separable, al agregarle más dimensión. Hay distintos tipos:

- Kernel lineal: Un núcleo lineal se puede utilizar como producto escalar normal de dos observaciones dadas cualesquiera. El producto entre dos vectores es la suma de la multiplicación de cada par de valores de entrada.

$$K(x, xi) = sum(x * xi)$$

- Kernel polinomial: Un núcleo polinomial es una forma más generalizada del núcleo lineal. El kernel polinomial puede distinguir el espacio de entrada curvo o no lineal.

$$K(x, xi) = 1 + sum(x * xi)^d$$

donde d es el grado del polinomio. d=1 es similar a la transformación lineal. El grado debe especificarse manualmente en el algoritmo de aprendizaje.

- Kernel de función de base radial (RBF): El *kernel* de función de base radial es una función de núcleo popular comúnmente utilizada en la clasificación. RBF puede mapear un espacio de entrada en un espacio dimensional infinito.

$$K(x, xi) = \exp(-gamma * sum((x - xi)^2$$

Aquí, gamma es un parámetro, que varía de 0 a 1. Un valor más alto de gamma se ajustará perfectamente al conjunto de datos de entrenamiento, lo que provoca un ajuste excesivo. Gamma=0.1 se considera un buen valor predeterminado. El valor de gamma debe especificarse manualmente en el algoritmo de aprendizaje.

Para más detalles, se sugiere consultar los sitios datacamp.com[4] y scikit-learn.org[5].

7.2.3. Red Neuronal

Para este modelo se aplica la API secuencial de Keras, una libreria de alto nivel que utiliza la librería *tensorflow p*ara realizar las operaciones fundamentales necesarias en un algoritmo de red neuronal.

[4] https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
[5] https://scikit-learn.org/stable/modules/svm.html

Esta aplicación nos permite armar una red neuronal muy simple utilizando solo capas totalmente conectadas una detrás de otra. En la Figura 5 podemos ver una red de 3 capas, cada una compuesta de nodos, a su vez, los nodos de la capa de *input* (entrada) representan las etiquetas (*features),* del *dataset* de estudio, el output (salida), el resultado obtenido. Cada neurona tiene un conjunto de pesos que deben mantenerse. Un peso para cada conexión de entrada y un peso adicional para un *bias*.

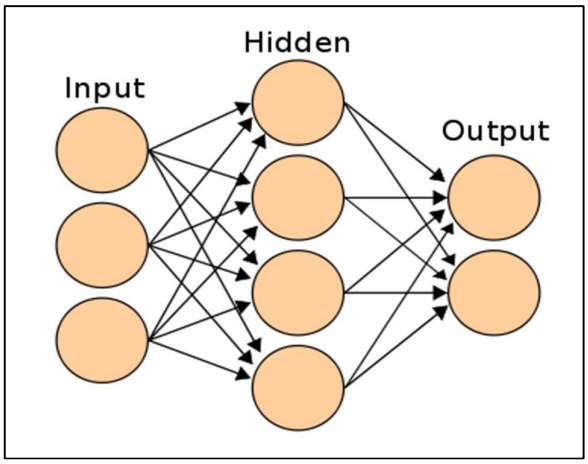

Fig. 5 – Red neuronal de tres capas
FUENTE

En el inicio el algoritmo configura de manera aleatoria valores arbitrarios a los pesos de cada conexión entre neuronas, luego simultáneamente realiza una sumatoria de la multiplicación de los pesos por los valores provenientes de la capa anterior, sumando un *"bias"*, luego para romper con la dependencia lineal del resultado obtenido en la sumatoria mencionada se aplica lo que se denomina una función de activación a la salida de cada neurona, este proceso se repite en cada capa hasta llegar a la salida de la red, este proceso generalmente es llamado *feed forward* [6].

Una vez obtenido el resultado se aplica una función de coste desde la salida de la red hasta la capa de input, es decir, va al sentido contrario del *feed forward*, esta función de coste conjuntamente con sus respectivas derivadas en cada capa detecta las conexiones que tienen un desvió mayor y necesitan ser ajustadas, con esto luego se ajustan los pesos teniendo en cuenta un hiperparámetro llamado *learning ratio*, todo

este proceso que ocurre después del *feed forward* se denomina comúnmente *back propagation* o descenso del gradiente [6].

Tanto el *feed forward* como el *back propagation* se ejecutan hasta disminuir el error hasta un parámetro aceptable según la problemática abordada. Se pueden consultar más detalles en la documentación de Keras[6]

7.3. Criterios de Comparación

Los criterios que hemos definido para el posterior análisis de los modelos, con el objetivo de determinar umbrales de probabilidad óptimos, serán en principio la matriz de confusión, la curva ROC (Receiver Operating Characteristic) y la curva PR (Precision-Recall).

Precision (precisión), representa el ratio de correctos positivos que el modelo predice. El resultado se encuentra entre 0 (no predice resultados positivos), y 1 (la totalidad de casos positivos que el modelo encuentra son correctos).

$$Precision = \frac{TP}{TP + FP}$$

Recall (exhaustividad), representa la cantidad de positivos que el modelo predice considerando la totalidad de positivos que se encuentran en el *dataset*. Este valor oscila entre 0 (el caso de que el modelo no encuentre verdaderos positivos), y 1 (el modelo encontró todos los casos positivos que se encuentran en el *dataset*).

$$Recall = \frac{TP}{TP + FN}$$

F1 representa una combinación de ambos ratios anteriores en un solo valor y nos permite comprender de manera más definida la performance de un modelo en dónde los datos no se encuentran balanceados.

$$F1 = 2\,\frac{Recall * Precision}{Recall + Precisioón}$$

[6] https://keras.io/api/models/sequential/

Por otro lado, la matriz de confusión nos muestra de manera integrada los falsos positivos, los positivos reales, los falsos negativos y los negativos reales que el modelo ha generado. Es una representación sucinta del resultado.

La curva ROC representa el balanceo entre los positivos reales y los falsos positivos, mientras que la curva PR refiere a la relación entre los positivos reales y los positivos totales utilizando diferentes umbrales de probabilidad. En la Figuras 6 y 7, se puede ver un ejemplo de curva ROC y curva PR, respectivamente.

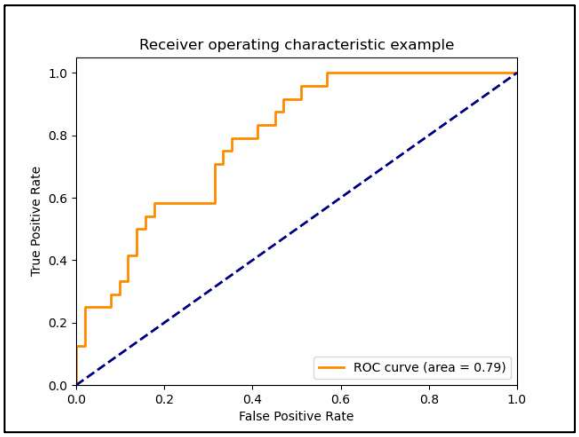

Figura 6 – Curva ROC

Fuente: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

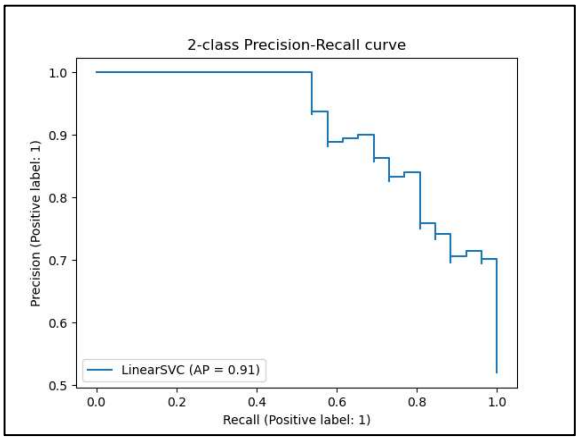

Figura 7 – Curva Precision Recall

Fuente: https://scikit-learn.org/stable/_images/sphx_glr_plot_precision_recall_001.png

En general, la opción PR resulta más aplicable a set de datos desbalanceados, es decir, donde las clases a determinar no se encuentran en una paridad numérica En nuestro caso, encontramos muchos registros negativos y muy pocos positivos (“A”), por lo cual utilizaremos la matriz de confusión y la curva PR para comparar los métodos [1][4].

8. Desarrollo del Modelo

8.1. Fuentes de Información

Como fuente de datos, se cuenta con un set de datos provistos por la entidad bancaria, y una etiqueta generada por NOSIS, obtenido por la junta de ambos reportes a través del campo “DNI”.

Las variables que lo componen son de la más diversa variedad, tal como se muestra a continuación:

- CUIT -
- NIVEL_SOCIO_ECONOMICO: determinado por el Banco (A,B,C1,C2,C3,D1,D2,D3)
- SCORE: Refiere a la etiqueta de NOSIS
- Cantidad de Convenios: supone los convenios de Plan Sueldo o Acreditaciones que puede poseer una persona
- Sucursal del Convenio: sucursal que administra la relación.
- N° DNI
- Codigo Postal
- Saldo del mes anterior: saldo de caja de ahorros al cierre del mes
- Acreditaciones de Sueldo: ingreso neto acreditado en cuenta
- Otras Acreditaciones: otras acreditaciones que no ingresan por codificación de acreditación de haberes.
- Acreditación de Plazo Fijo: plazos fijos cobrados.
- Pago de Préstamos: pago de cuota de préstamos vigentes del mes.
- Pago de Tarjeta de Crédito: pago al vencimiento de todas la tarjetas de crédito con saldo durante el mes
- Compras con Tarjeta de Débito: compras del mes con tarjeta de débito

- Débitos Directos: débitos automáticos en cuenta caja de ahorros.
- Débito de Mutuales: pago a mutuales por servicios prestados durante el mes.
- Pagos de Servicios Link: pago de servicios a través de cajeros automáticos y home banking.
- Extracciones por ATMs: extracciones por cajeros automáticos.
- Extracciones por Caja Humana: extracciones por caja.
- Constitución de Plazo Fijo: plazo fijo constituido.
- Transferencias Realizadas: transferencias realizadas por canales (no incluye MEP).
- Compra de Moneda Extranjera: compra de moneda extranjera por canales automáticos (no incluye otras operaciones cursadas por la Oficina de Comercio Exterior).
- Tarjeta de Crédito Tuya Tipo: el concepto “Tipo” incluye varios estados, a saber:
    - Comedores: refrigerio organismos públicos
    - Tradicional: es la tarjeta de crédito común segmento individuos
    - Empresa: es la tarjeta para empresas
    - Adicional: refiere a adicionales
    - Recargable: son utilizadas para algunos complementos salariales
    - Pago Fijo: clientes con pago mínimo fijo en la tarjeta

- Tarjeta de Crédito Tuya Estado: Inicial (es la emitida) / Operativa (activada por el cliente)
- Tarjeta de Crédito Tuya Saldo en Pesos: saldo a pagar en pesos del resumen
- Tarjeta de Crédito Tuya Saldo en Dólares: saldo a pagar en dólares del resumen
- Tarjeta de Crédito Visa Tipo: registra los siguientes tipos de plástico (segmentados a partir del límite de compra asignado), a saber:
    - VISA PENDIENTE: es la tarjeta en tránsito
    - VISA NACIONAL: tarjeta de crédito de uso en el país solamente
    - SIGNATURE: tarjeta de crédito segmento de altos ingresos
    - PLATINUM: es la más selecta de las tarjetas
    - VISA INTERNACIONAL: corresponde a segmento de ingresos medios
    - VISA PREMIUM ORO: ingresos medios altos

- Tarjeta de Crédito Visa Estado: registra los siguientes estados:
    - Inicial: es la emitida antes de ser recibida por el cliente
    - Operativa: es la que se encuentra en poder del cliente
    - Operativa en Mora: es la que se encuentra con atraso de más de 60 días

- Tarjeta de Crédito Visa Saldo en Pesos: saldo a pagar en pesos en el resumen
- Tarjeta de Crédito Visa Saldo en Dólares: saldo a pagar en dólares en el resumen
- Tarjeta de Crédito MasterCard Tipo: registra la siguiente tipología:
    - PLATINUM MC
    - MASTERCARD BLACK
    - MC EXECUTIVE CORPORATE
    - MC REGIONAL
    - GOLD MC
    - MC INTERNACIONAL
    - CORPORATE MC INTERNACIONAL
- Tarjeta de Crédito MasterCard Estado: Inicial (es la emitida) / Operativa (emitida en poder del cliente)
- Tarjeta de Crédito MasterCard Saldo en Pesos: saldo a pagar en pesos en el resumen
- Tarjeta de Crédito MasterCard Saldo en Dólares: saldo a pagar en dólares en el resumen
- Tarjeta de Crédito Cabal Tipo: “Cabal Internacional” es el único tipo registrado
- Tarjeta de Crédito Cabal Estado: “Operativa” es el único estado disponible
- Tarjeta de Crédito Cabal Saldo en Pesos: saldo a pagar en pesos en el resumen
- Tarjeta de Crédito Cabal Saldo en Dólares: saldo a pagar en dólares en el resumen
- Plazo Fijo Vigente Pesos: plazo fijo activo en el sistema
- Plazo Fijo Vigente Dólares: plazo fijo en dólares activo en el sistema
- Plazo Fijo Vigente Euros: plazo fijo en euros vigente en el sistema
- Préstamos vigente línea 1XX: préstamos personales hasta 36 meses
- Préstamos vigente línea 4XX: préstamos personales hasta 60 meses
- Préstamos vigente línea 5XX: préstamos personales de anticipos de sueldo hasta 180 días de plazo
- Cuenta Corriente Saldo: saldo en cuenta corriente en pesos
- Caja de Seguridad: caja de seguridad por titular de la misma
- Tarjeta de Débito: registra 2 estados:
    - Habilitada
    - Inhabilitada
- Home Banking uso: refiere al ingreso al home banking en el mes
- ATM de mayor frecuencia: refiere al cajero automatico más usado

- Fecha de Nacimiento
- Jurisdicción del Convenio: refiere a las 180 jurisdicciones registradas en la administración pública (clientes "Plan Sueldos" pertenecientes al estado provincial)
- Visa Recargable: tipo de tarjeta VISA similar a "Tuya Recargable"
- MasterCard Recargable: tipo de tarjeta MasterCard similar a "Tuya Recargable"

La información generada por NOSIS, por otro lado, tiene que ver con la segmentación por nivel socioeconómico que realiza. En la Figura 8 podemos ver dicha segmentación.

Como se indicó en la Sección 3, resaltamos que el Banco utiliza el mismo criterio de segmentación que realiza NOSIS, en cuanto a criterios de ingresos, pero la segmentación que realiza el Banco sólo considera el salario líquido ingresado en la cuenta y NOSIS utiliza todas las fuentes de información disponible (fiscales, laborales y crediticias), que pueden estar enriquecidas con otras fuentes no tradicionales (por ej., supermercados).

NSE	%	Ingreso Estimado Junio-2021		
		Mínimo	Máximo	Promedio
A	5.00%	298,762	-	461,701
B	10.00%	186,040	298,761	228,759
C1	30.00%	95,334	186,039	128,951
C2	25.00%	66,606	95,333	79,844
C3	15.00%	47,017	66,605	57,519
D1	10.00%	30,099	47,016	38,625
D2	5.00%	-	30,098	23,212
Total	**100.00%**	**-**	**-**	**118,258**

Fig. 8 – Segmentación de nivel socioeconómico realizada por NOSIS

Del *dataset* para realizar "training" y "testing", la etiqueta respecto al segmento socioeconómico refiere a NOSIS propiamente dicho, asimismo la cantidad de casos se encuentra acotada a los clientes respecto de los cuales se haya formulado algún

análisis crediticio (o pedidos de informe), o hayan tomado alguna línea de crédito por canales automáticos (en tal caso, se dispara automáticamente una solicitud para realizar el “scoring” y asignar el monto).

8.2. Proceso de ETL Aplicado

Luego de una caracterización inicial del dataset, fueron eliminados los campos CUIT, SCORE, N° DNI, Código Postal, Sucursal del Convenio. Estos campos fueron removidos debido a que sus etiquetas no conforman información cualitativa respecto al modelo (puntualmente los campos CUIT, N° DNI). En el caso de los campos Código Postal y Sucursal del Convenio, del análisis exploratorio se tomó conocimiento de que uno de los principales problemas con el sistema de altas masivas tiene que ver con el hecho de que el domicilio asignado frecuentemente resulta inexacto, situación que conlleva a que la sucursal asignada contiene la misma inexactitud (es decir, personas que son asignadas a una sucursal que no tiene proximidad a su domicilio real).

De esta forma, el set de datos definitivo, sobre el cual se validarán los modelos predictivos, cuenta con 37763 registros y 51 campos.

Como bien se pudo apreciar en la Sección 8.1, algunas variables están representadas por números y otras refieren a otros criterios de clasificación, las que finalmente requieren un proceso de transformación para proceder a su análisis. Por tal motivo, se procedió a llenar todos los espacios con valores nulos para que puedan ser convertidos a variables categóricas 1 o 0 respecto a aquellas categorías que necesitaban ser modificadas.

Asimismo, la fecha de nacimiento se convirtió a tipo *“numérico”* para la misma finalidad.

Para programar este ETL, se utilizó el paquete *“get.dummies”* de la librería pandas.

En el Anexo 12.1. se puede consultar el código fuente del ETL implementado.

8.3. Implementación de los Modelos

En todos los casos se determinaron umbrales de *"training"* y "testing" del 75% y 25%, los mismos fueron determinados sobre criterios recomendados por la bibliografía consultada. De las pruebas exploratorias previas realizadas (con umbrales diferentes, nunca training inferior al 65%), no se observaron cambios significativos en los resultados del modelo.

Como lenguaje para codificar la transformación de los datos y los algoritmos de modelado, se utilizó Python, junto con las librerías Pandas y Sklearn.

8.3.1. Modelo Basado en Lightgbm

Para la construcción del modelo basado en Lightgbm, se utilizó el paquete StandardScaler de la librería Sklearn.

El modelo se ajustó considerando los siguientes parámetros: learning_rate = 0.003, boosting_type = 'gbdt', objective = 'binary', metric = 'binary_logloss', sub_feature = 0.5, num_leaves = 10, min_data = 50, max_depth = 15.

En las Figuras 9, 10 y 11 se pueden ver la matriz de confusión normalizada y sin normalizar, y la curva PR obtenidas con este modelo, respectivamente.

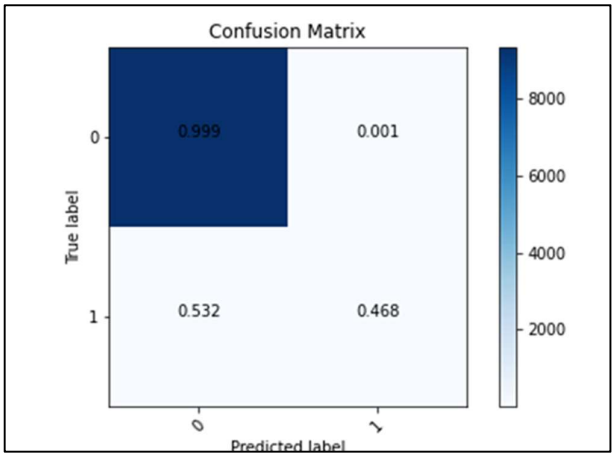

Fig. 9 - Matriz de confusión normalizada para Lightgbm

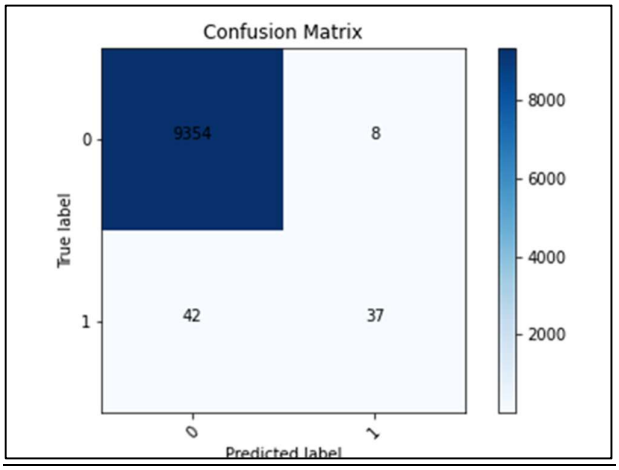

Fig. 10 - Matriz de confusión sin normalizar para Lightgbm

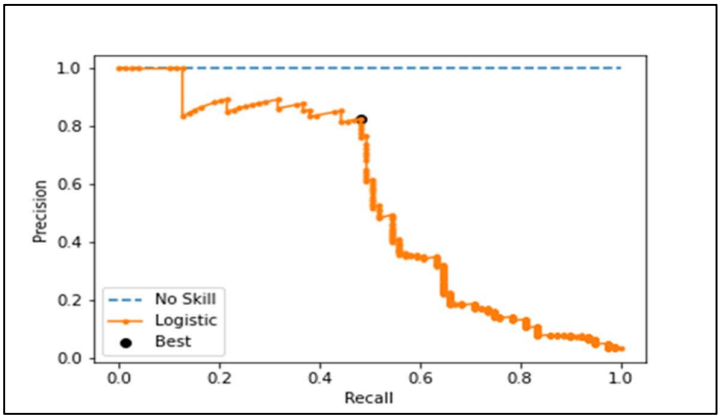

Fig. 11 – Curva PR para Lightgbm

Dado que se cuenta con un dataset muy desbalanceado el algoritmo clasifica todas las entradas como una sola clase, en este caso, la clase 0. Una de las soluciones analizadas para tratar en este tipo de circunstancias es modificar el umbral con el que se clasifica una clase como 1 o 0, normalmente el umbral utilizado es 0.5, si la salida del algoritmo es mayor a 0.5 se toma como clase 1 sino como clase 0.

Para encontrar el umbral óptimo se utilizó la curva *precision-recall*, la cual muestra el balance entre las medidas de *precision* y *recall* para cada umbral, asimismo para determinar el mejor umbral se calcula un indicador llamado *f-score*, para cada *f-score*

existe un umbral asociado, entonces una vez se obtiene el *f-score* más alto encontramos el umbral óptimo para clasificar las salidas del algoritmo.

En la gráfica de la Figura 11 se puede observar el punto que representa los valores de *precision* y *recall* para los cuales se obtiene el *f-score* más alto y, por lo tanto, el umbral óptimo. Para este caso se obtuvo un f-score de 0.608 que le corresponde un umbral de 0.208548.

En el Anexo 12.2. se puede consultar el código fuente.

8.3.2. Modelo basado en Neural Network

Para este modelo, se definieron dos capas intermedias y una capa final de salida, a modo de definir umbrales de probabilidad. Se codificó una red neuronal interconectada con 2 capas intermedias (de 16 y 32 neuronas respectivamente), y una capa de salida:

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 16)	6528
dense_13 (Dense)	(None, 32)	544
dense_14 (Dense)	(None, 1)	33

Total params: 7,105
Trainable params: 7,105
Non-trainable params: 0

En las Figuras 12 y 13 se pueden ver la matriz de confusión normalizada y sin normalizar, respectivamente.

En la Figura 14 se puede ver la curva Precision-Recall, sobrel o cual quedó configurado el balanceo del modelo. En base a esto, se estimó el umbral de posibilidad óptimo en 31,19%.

En el Anexo 12.3. se puede consultar el código fuente.

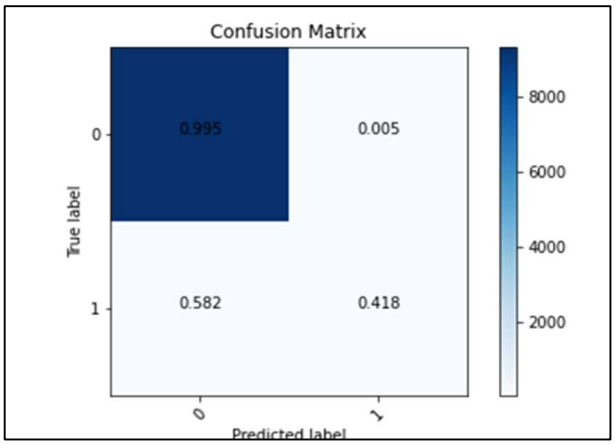

Fig. 12 - Matriz de confusión normalizada poara RN

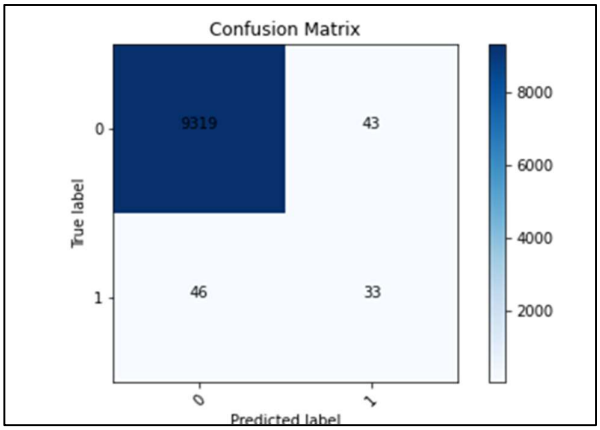

Fig. 13 - Matriz de confusión sin normalizar para RN

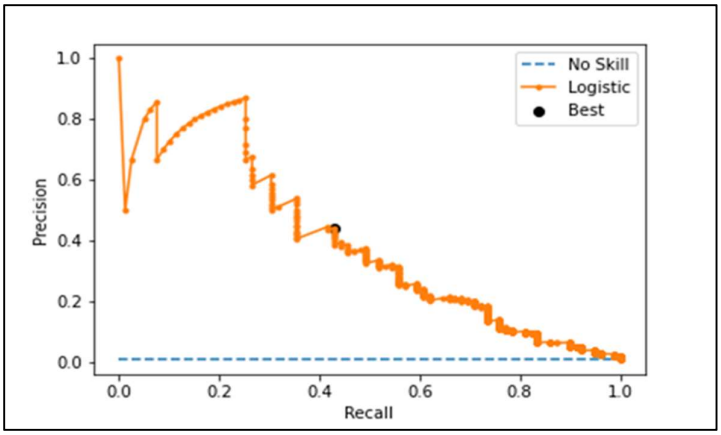

Fig. 14 – Curva Precisión-Recall para Neural Network

8.3.3. Support Vector Machine

En el caso de “Support Vector Machine” se ensayaron diferentes parámetros para ajustar una salida aceptable. Realizar en una sola línea de código una suerte optimización bayesiana resultó imposible, dada la demanda de capacidad de procesamiento. No obstante, se procedió a implementar un set de parámetros que resultaron más que aceptables en razón a la performance exhibida.

El modelo kernel utilizado fue “Radial Basic Function”, con parámetros C=1, Gamma=1 de la librería *“Sklearn”*. La matriz de confusión resultante normalizada puede verse en la Figura 15, y en la Figura16 se la puede ver sin normalizar.

En el Anexo 12.4. se puede consultar el código fuente.

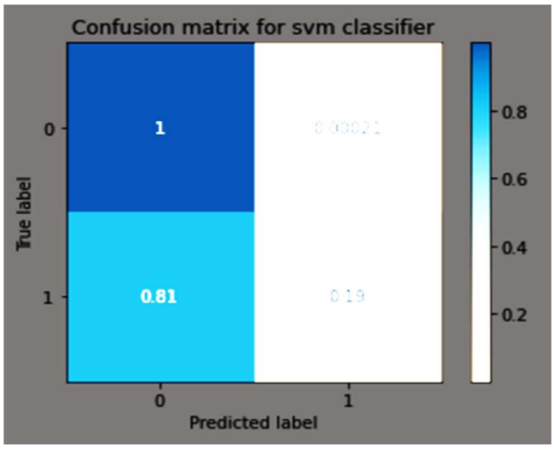

Fig. 15 - Matriz de confusión normalizada SVM

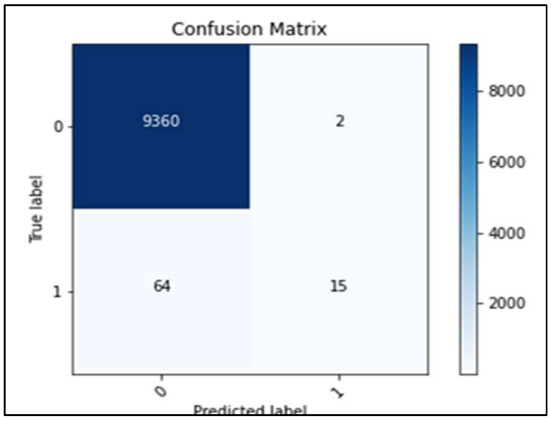

Fig. 16 – Matriz de confusión sin normalizar para SVM

9. Resultados Experimentales

En general los 3 modelos exhibieron desempeños más que aceptables, como puede verse en la siguiente tabla:

Modelo	Accuracy	Precision	Recall	F1	Best PR	Tiempo de Entrenamiento (seg)
Lightgbm	0.9947	0.8222	0.4683	0.608	0.208548	528
Red Neuronal	0.99057	0.4342	0.4177	0.436	0.311986	135
SVM	0.9930	0.8823	0.1898			326

Support Vector Machine fue el que mejor desempeño demostró, pero no está exento del posible "overfitting". En tal sentido el algoritmo mejor balanceado sería el de Redes Neuronales.

Esto último tiene que ver con la precisión que exhibió el modelo, el cual se aplica sobre un set de datos que corresponde a un solo mes en curso (junio 2021), es decir, sólo se evaluaron en las matrices respectivas los denominados *"training errors"*. Al no ensayarse sobre otros meses no habría que descartar que la salida del modelo se encuentre en exceso ajustada a la muestra sobre la cual procede el ensayo y el mismo exhiba dificultades para ajustarse a otros espacios *"muestrales"* no observados, ni analizados en este trabajo (*"generalization errors"*).

Esto a su vez es lo que se denomina *"overfitting"*, que en otras palabras denota la dificultad de expandir las conclusiones de los resultados en los ensayos hacia una generalización que permita su aplicabilidad sobre un objeto de estudio en particular. El modelo se encuentra muy ajustado a la muestra sobre la cual procede su análisis [2].

10. Conclusiones y Trabajo a Futuro

En base a los resultados obtenidos, podemos concluir que, el análisis exploratorio sobre una base de datos que permita generar alguna salida útil para la toma de decisiones, ha sido satisfactorio. Se estudiaron algoritmos que resultaron sumamente útiles para determinar una segmentación socio-económica conforme los parámetros de una base de datos enriquecida por sus distintas y variadas fuentes, ajena a la entidad (NOSIS).

Las salidas del modelo resultaron suficientes para cualquier modelo de negocio que quiera diseñarse con algún criterio de segmentación de esta índole (que para nada es el único ni excluyente).

De todas maneras, no escapa a las conclusiones de este trabajo la necesidad de continuar con el análisis de optimización de los conjuntos de variables e incorporar un espacio temporal más amplio. que le otorgue mayor consistencia a las salidas del modelo. Elegir el modelo más apropiado también constituye un desafío no menor, dado que en el mismo habría que ponderar las complejidades del gobierno de datos de la entidad y las posibilidades que ofrecen los modelos en términos predictivos o de clasificación (como refiere el presente caso). En este sentido, se entiende que deberán ser ponderadas las posibilidades que ofrecen los algoritmos y, probablemente, posterior a una selección de los mismos, capturar los *“bemoles”* del oficio, para especializarse en el análisis exploratorio de datos.

11. Referencias

[1] Branco, P., Torgo, L., Ribeiro, R. (2016). *A Survey of Predictive Modeling on Imbalanced Domains*. ACM Computing Surveys, 48 (2), Article No.: 31, pp 1–50. https://doi.org/10.1145/2907070

[2] Chiu, S. & Tavella, D. (2008). *Introduction to Data Mining*. *Data Min. Mark. Intell. Optim. Mark. Returns*, pp. 137–192, 2008, doi: 10.1016/b978-0-7506-8234-3.00007-1. – Págs 172-174

[3] Hastie, T. et. all. (2009). The Elements of Statistical Learning }. Springer Series in Statistics. *Math. Intell.*, vol. 27, no. 2, pp. 83–85, 2009.

[4] Juba, B., & Le, H. S. (2019). Precision-Recall versus Accuracy and the Role of Large Data Sets. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4039-4048. https://doi.org/10.1609/aaai.v33i01.33014039

[5] Ke, G. et al (2017). LightGBM: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.*, vol. 2017-December, no. Nips, pp. 3147–3155, 2017.

[6][B] Mützel, S. (2015). *Facing Big Data: Making sociology relevant*. *Big Data Soc.*, vol. 2, no. 2, pp. 1–4, 2015, doi: 10.1177/2053951715599179.

12. Anexo

12.1. Código del ETL

```
#importo las librerías que voy a utilizar
 import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importo el dataset del postgres
dataset = pd.read_csv(r"C:\Users\Sebastian\Desktop\Itba\Trabajo Final\nosis1.csv")

#Determino las columnas a eliminar:
dataset.drop(['CUIT','Sucursal del Convenio','SCORE','N° DNI','Codigo Postal'],axis=1,inplace=True)

#1er paso: identificar los valores correspondientes a las columnas no numéricas:
dataset['NIVEL_SOCIO_ECONOMICO'].unique()
dataset['Tarjeta de Crédito Tuya Tipo'].unique()
dataset['Tarjeta de Crédito Tuya Estado'].unique()
dataset['Tarjeta de Crédito Visa Tipo'].unique()
dataset['Tarjeta de Crédito Visa Estado'].unique()
dataset['Tarjeta de Crédito MasterCard Tipo'].unique()
dataset['Tarjeta de Crédito MasterCard Estado'].unique()
dataset['Tarjeta de Crédito Cabal Tipo'].unique()
dataset['Tarjeta de Crédito Cabal Estado'].unique()
dataset['Caja de Seguridad'].unique()
dataset['Tarjeta de Débito'].unique()
dataset['Home Banking uso'].unique()
dataset['ATM de mayor frecuencia'].unique()
dataset['Fecha de Nacimiento'].unique()
dataset['Jurisdicción del Convenio'].unique()
dataset['Visa Recargable'].unique()
dataset['MasterCard Recargable'].unique()
dataset['Tuya Recargable'].unique()
```

```
dataset['Cabal Recargable'].unique()

#verificar (nulos), NAs e inserto "_" para la posterior identificación del campo sin valor:
dataset['Tarjeta de Crédito Tuya Tipo'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito Tuya Estado'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito Visa Tipo'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito Visa Estado'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito MasterCard Tipo'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito MasterCard Estado'].replace(' ','_',regex=True,
inplace=True)
dataset['Tarjeta de Crédito Cabal Tipo'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Crédito Cabal Estado'].replace(' ','_',regex=True, inplace=True)
dataset['Caja de Seguridad'].replace(' ','_',regex=True, inplace=True)
dataset['Tarjeta de Débito'].replace(' ','_',regex=True,inplace=True)
dataset['Home Banking uso'].replace(' ','_',regex=True, inplace=True)
dataset['ATM de mayor frecuencia'].replace(' ','_',regex=True, inplace=True)
dataset['Jurisdicción del Convenio'].replace(' ','_',regex=True, inplace=True)
dataset['Visa Recargable'].replace(' ','_',regex=True, inplace=True)
dataset['Tuya Recargable'].replace(' ','_',regex=True, inplace=True)
dataset['MasterCard Recargable'].replace(' ','_',regex=True, inplace=True)
dataset['Cabal Recargable'].replace(' ','_',regex=True, inplace=True)

#defino las variables dependientes e independientes, elimino la etiqueta a identificar:
X=dataset.drop('NIVEL_SOCIO_ECONOMICO',axis=1).copy()
y=dataset['NIVEL_SOCIO_ECONOMICO'].copy()

#creo una function que asigne 1 y 0 a la etiqueta a predecir:
def NIVEL_SOCIO_ECONOMICO(val):

    if val == 'A':
        return 1
    else:
        return 0

#aplico la función a la etiqueta "y":
```

```
y=y.apply(NIVEL_SOCIO_ECONOMICO)

#”One hot encoding” para clasificar las variables (crea un campo por cada categoría en las columnas no numéricas:
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Tuya Tipo'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Tuya Estado'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Visa Tipo'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Visa Estado'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito MasterCard Tipo'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito MasterCard Estado'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Cabal Tipo'])
X=pd.get_dummies(X, columns=['Tarjeta de Crédito Cabal Estado'])
X=pd.get_dummies(X, columns=['Caja de Seguridad'])
X=pd.get_dummies(X, columns=['Tarjeta de Débito'])
X=pd.get_dummies(X, columns=['Home Banking uso'])
X=pd.get_dummies(X, columns=['ATM de mayor frecuencia'])
X=pd.get_dummies(X, columns=['Jurisdicción del Convenio'])
X=pd.get_dummies(X, columns=['Tuya Recargable'])
X=pd.get_dummies(X, columns=['Visa Recargable'])
X=pd.get_dummies(X, columns=['MasterCard Recargable'])
X=pd.get_dummies(X, columns=['Cabal Recargable'])

#considero la fechad de nacimiento como valor numérico
X['Fecha de Nacimiento']=pd.to_datetime(X['Fecha de Nacimiento']).dt.strftime("%d%m%Y")
X['Fecha de Nacimiento'].unique()
X['Fecha de Nacimiento'].astype(int)
```

12.2. Código del Modelo Lightgbm

```
#importo las librerías:
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import confusion_matrix, precision_score
import matplotlib.pyplot as plt
import pandas as pd
import lightgbm as lgb
import numpy as np
import itertools
from numpy import argmax

# función para plot de matriz de confusión:
def plot_confusion_matrix(cm,
                          classes,
                          normalize=False,
                          title='Confusion Matrix',
                          name_for_saving = 'Confusion Matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45 )
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print('Normalized Confusion Matrix')
    else:
        print('Confusion matrix without normalization')

    print(cm)
```

```
    thresh = cm.max()/2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        plt.text(j, i, cm[i, j],
                 horizontalalignment='center',
                 color='white' if cm[i,j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(f'{name_for_saving}.png')

#importo los dasets:
X = pd.read_csv('data/nosis_final.csv')
y = pd.read_csv('data/labels.csv')

#separo training y testing:
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)

#normalizo el dataset:
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

#genero el modelo:
d_train = lgb.Dataset(x_train, label=y_train)

#seteo del clasificador de lightgbm:
params = {}
params['learning_rate'] = 0.003
params['boosting_type'] = 'gbdt'
params['objective'] = 'binary'
params['metric'] = 'binary_logloss'
```

```
params['sub_feature'] = 0.5
params['num_leaves'] = 10
params['min_data'] = 50
params['max_depth'] = 15

#entreno el modelo:
clf = lgb.train(params, d_train, 1000)

# Aplico:
y_pred=clf.predict(x_test)

# calculo pr-curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred)

# plot the pr curve for the model
no_skill = len(y_test[y_test==1]) / len(y_test)
plt.plot([0,1], [no_skill,no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision, marker='.', label='Logistic')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
# show the plot
plt.show()

# convert to f score
fscore = (2 * precision * recall) / (precision + recall)
# locate the index of the largest f score
ix = argmax(fscore)
print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix], fscore[ix]))

# plot the roc curve for the model
no_skill = len(y_test[y_test==1]) / len(y_test)
plt.plot([0,1], [no_skill,no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision, marker='.', label='Logistic')
```

```
plt.scatter(recall[ix], precision[ix], marker='o', color='black', label='Best')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
# show the plot
plt.show()

#argmax() method – determino el umbral y aplico al testing:
y_pred_rounded = [1 if i >= 0.208548 else 0 for i in y_pred]

#using precision score for error metrics
precision_score(y_pred_rounded ,y_test,average=None).mean()
cm = confusion_matrix(y_true=y_test, y_pred= y_pred_rounded)
cm_plot_labels = ['0', '1']

# plot de matriz sin normalizar
plot_confusion_matrix(cm=cm,
                      classes=cm_plot_labels,
                      title='Confusion Matrix',

name_for_saving='conf_matrix_threshold_moving_precision_recall_curve')

# plot de matriz normalizada
plot_confusion_matrix(cm=cm,
                      classes=cm_plot_labels,
                      normalize=True,
                      title='Confusion Matrix',

name_for_saving='conf_matrix_threshold_moving_precision_recall_curve')

TP , TN, FP, FN = 37, 9354, 8, 42

accuracy = (TP + TN)/(TP + TN + FP + FN)
accuracy
```

recall = TP /(TP + FN)
recall

precision = TP / (TP + FP)
precisión

12.3. Código del Modelo Red Neuronal

```
#importo las librerías que voy a usar:
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import numpy as np
from random import randint
from sklearn.utils import shuffle
from sklearn.preprocessing import MinMaxScaler
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from sklearn.metrics import confusion_matrix
import itertools
import matplotlib
from matplotlib import pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
# Feature Scaling
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import precision_score
from numpy import argmax

# función para plot de matriz de confusión
def plot_confusion_matrix(cm,
                          classes,
                          normalize=False,
                          title='Confusion Matrix',
                          name_for_saving = None,
                          cmap=plt.cm.Blues):


    """
    This function prints and plots the confution matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45 )
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print('Normalized Confusion Matrix')
    else:
        print('Confusion matrix without normalization')

    print(cm)

    thresh = cm.max()/2.
```

```
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

     plt.text(j, i, cm[i, j],
              horizontalalignment='center',
              color='white' if cm[i,j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig(f'{name_for_saving}.png')
dataset = pd.read_csv('data/nosis1.csv')
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)

#voy a escalar la información:
train_samples=np.array(x_train)
train_labels=np.array(y_train)
train_labels,train_samples=shuffle(train_labels,train_samples)
x_val=np.array(x_test)
y_val=np.array(y_test)

scaler = MinMaxScaler(feature_range=(0,1))
scaled_train_samples = scaler.fit_transform(train_samples)
x_val=scaler.fit_transform(x_val)
valid_set=(x_val,y_val)

scaled_train_samples.shape, train_labels.shape, train_samples.shape
train_labels.reshape(-1, 1).shape, train_labels.shape
scaled_train_samples.shape[1]



# Build Model
# en “activation” se cambia softmax que es usado en clasificación multiclase
# por sigmoid que es usado pora una clasificación binaria
model = Sequential([
```

```
    Dense(units=16, input_shape=(scaled_train_samples.shape[1],)),
    Dense(units=32, activation='relu'),
    Dense(units=1, activation='sigmoid')
])

model.summary()

# en la compilación se cambia sparse_categorical_crossentropy por binary_crossentropy
model.compile(optimizer=Adam(learning_rate=0.003), loss='binary_crossentropy', metrics=['accuracy'])

model.fit(x=scaled_train_samples,
          y=train_labels,
          validation_data=valid_set,
          batch_size=32,
          epochs=10,
          verbose=2)

# test samples normalization
scaled_test_samples = scaler.fit_transform(x_test)

#predictions = model.predict(x= scaled_test_samples, batch_size=10, verbose=0)
predictions = model.predict(x= scaled_test_samples, batch_size=10, verbose=0)


#Precision Recall, calculo pr-curve
precision, recall, thresholds = precision_recall_curve(y_test, predictions)

# plot  pr-curve
no_skill = len(y_test[y_test==1]) / len(y_test)
plt.plot([0,1], [no_skill,no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision, marker='.', label='Logistic')
# axis labels
plt.xlabel('Recall')
```

```
plt.ylabel('Precision')
plt.legend()
# show the plot
plt.show()

# convert to f score
fscore = (2 * precision * recall) / (precision + recall)
# locate the index of the largest f score
ix = argmax(fscore)
print('Best Threshold=%f, F-Score=%.3f' % (thresholds[ix], fscore[ix]))

# plot the roc curve for the model
no_skill = len(y_test[y_test==1]) / len(y_test)
plt.plot([0,1], [no_skill,no_skill], linestyle='--', label='No Skill')
plt.plot(recall, precision, marker='.', label='Logistic')
plt.scatter(recall[ix], precision[ix], marker='o', color='black', label='Best')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
# show the plot
plt.show()

#argmax() method
y_pred_rounded = [1 if i >= 0.369596 else 0 for i in predictions] # este valor corresponde al umbral determinado

#using precision score for error metrics
precision_score(y_pred_rounded ,y_test,average=None).mean()

cm = confusion_matrix(y_true=y_test, y_pred= y_pred_rounded)

cm_plot_labels = ['0', '1']

# plot de matriz de confusión sin normalizar
```

```
plot_confusion_matrix(cm=cm,
                      classes=cm_plot_labels,
                      title='Confusion Matrix',
                      name_for_saving='conf_matrix_tensorflow_binary_precision_recall')

# matriz de confusión normalizada
plot_confusion_matrix(cm=cm,
                      classes=cm_plot_labels,
                      normalize=True,
                      title='Confusion Matrix',
                      name_for_saving='conf_matrix_tensorflow_binary_precision_recall')
```

12.4. Código del Modelo SVM

```
#importo librerías:
import itertools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.decomposition import PCA
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, precision_score
from sklearn import metrics
```

```
# defino función plot de matriz de confusión
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion Matrix', cmap=plt.cm.Blues):

    """
    This function prints and plots the confution matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45 )
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print('Normalized Confusion Matrix')
    else:
        print('Confusion matrix without normalization')

    print(cm)

    thresh = cm.max()/2.

    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

        plt.text(j, i, cm[i, j],
                 horizontalalignment='center',
                 color='white' if cm[i,j] > thresh else 'black')

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.savefig('confusion_matrix_svm_classifier.png')
```

```
X = pd.read_csv('data/X_nosis_cleaned_in_tf_file.csv')
y = pd.read_csv('data/labels_nosis_cleaned_in_tf_file.csv')

#separo training and testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
0)
y_test.values.ravel()

# Feature Scaling
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

clf = svm.SVC(kernel='rbf')
clf.fit(x_train, y_train.values.ravel())

clf.score(x_train, y_train)

y_pred = clf.predict(x_test)

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))

matrix = plot_confusion_matrix(clf, x_test, y_test,
                               cmap=plt.cm.Blues,
                               normalize='true')
plt.title('Confusion matrix for svm classifier')
plt.show(matrix)
```

```
plt.show()

precision_score(y_pred,y_test,average=None).mean()
cm = confusion_matrix(y_test, y_pred)
cm_plot_labels = ['0', '1']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')

#optimización de parámetros
param_grid=[
   {'C':[0.5,1,10,100],
    'gamma':['scale',1,0.1,0.01,0.001,0.0001],
    'kernel':['rbf']},]

optimal_params=GridSearchCV(SVC(),
                            param_grid,
                            cv=5,
                            scoring='accuracy',
                            verbose=0)

optimal_params.fit(x_train,y_train.values.ravel())
optimal_params.best_estimator
# Get support vectors
support_vectors = clf.support_vectors_

# Visualize support vectors
plt.scatter(X_train[:,0], X_train[:,1])
plt.scatter(support_vectors[:,0], support_vectors[:,1], color='red')
plt.title('Linearly separable data with support vectors')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

# Plot decision boundary
plot_decision_regions(X_test, y_test, clf=clf, legend=2)
plt.show()
```