UNIVERSIDAD
TORCUATO DI TELLA

MASTER IN MANAGEMENT & ANALYTICS

Predicción de la Satisfacción de los Clientes en la Atención a partir de Características Conversacionales

Alejandro Romanisio

Mayo 2022

Tutor: Agustín Gravano

Resumen

Los servicios de atención al cliente son parte esencial de los elementos que hacen a la experiencia de los clientes. En este contexto, existe un valor muy grande en poder predecir cómo perciben los clientes la experiencia de los servicios de atención al cliente. Esto permitiría a las empresas aprender buenas prácticas en el manejo conversacional para generar mejores experiencias y también detectar posibles malas experiencias para revertirlas. Con esa motivación, en este trabajo se buscó entender las palancas que mueven esa experiencia y armar un modelo predictivo que prediga la puntuación que pondrán los clientes a los agentes tras una conversación vía WhatsApp, al contactarse con el servicio de atención al cliente. Se buscó predecir la nota a partir de tres tipos de información: semántica (palabras utilizadas por clientes y agentes durante la conversación), conversacional (tiempos y formas de la conversación) y contextual (características de la situación del cliente al momento de contactarse con atención al cliente). Así, se entrenaron 14 modelos predictivos con los algoritmos *Random Forest* y *XGBoost* para obtener un modelo predictivo y un entendimiento de los principales componentes que explican la posibilidad de que un cliente puntúe su experiencia como un promotor. Además del modelo predictivo, los aprendizajes revelados a partir de explorar los datos y el funcionamiento del modelo cobran importancia para la implementación de mejores prácticas en la empresa estudiada.

Abstract

Customer services is a key element in the customer experience. In that context, the prediction of clients' perception of the customer service experience is of great value. It would let firms gain insights on conversation management best practices, leading to better experiences and also to detect and revert bad experiences. Given this motivation, this work seeks to understand the drivers of this experience and build a model that predicts clients' score on customer service WhatsApp conversations. Score prediction was sought based on three types of information: semantic (words used by clients and agents during conversation), conversational (conversation timing and patterns) and contextual (clients' characteristics at the moment of contact). Thereby, 14 predictive models where trained with Random Forest and XGBoost algorithms in order to obtain a predictive model and an understanding of the main drivers leading to the possibility of a client giving a promoter score. In addition to the predictive model, the insights gained during data exploratory analysis and the working of the data model are of importance for the implementation of best practices in the firm studied.

Adjunto a este trabajo se encuentran disponibles los archivos de códigos utilizados para su realización:

- *script_mensajes.py*
- *modelos.py*
- *unit_testing.py*

Índice

Índice de tablas

Índice de figuras

1. Introducción

1.1 Contexto

La experiencia de los clientes en el mundo Fintech se encuentra fuertemente determinada por los niveles de servicio. La experiencia con atención al cliente es parte fundamental para determinar estos niveles de servicio y resulta uno de los *touchpoints* clave para poder tornar clientes detractores en clientes promotores.

En este contexto, resulta relevante en el mundo de atención al cliente poder detectar con velocidad y precisión si un usuario que se contactó con los canales de atención al cliente de una Fintech tiene potencial de convertirse en un detractor luego del contacto y en tal caso, resulta también relevante tomar una acción de mejora en el momento del contacto o de forma posterior para evitar que el usuario termine siendo un detractor.

Cuando hablamos de detractores, hablamos de clientes que no puntúan con la máxima nota las interacciones con la empresa en los distintos puntos de contacto. Cuando hablamos de promotores, por el contrario, hablamos de aquellos clientes que puntúan con la máxima nota una interacción con cualquier punto de contacto con la empresa. Una de las interacciones más importantes es la que se da al momento de contacto con atención al cliente, ya que es un momento decisivo y de potencial *churn* dado que los contactos suelen darse cuando los clientes ya tuvieron una fricción con el producto o servicio ofrecido.

Existe una correlación positiva entre la proporción de promotores y detractores y la rentabilidad y crecimiento de una empresa, en particular en la industria financiera. Por ejemplo, los clientes promotores de bancos tienen saldos casi 45% superiores a los detractores; asimismo, la tasa de abandono de los detractores resulta ser el triple comparada a los promotores[1]. De esta forma, se encuentra una conexión directa entre experiencia de los clientes y rentabilidad y crecimiento del negocio.

Así, volviendo al contexto en análisis en este trabajo, tienen relevancia para la rentabilidad y crecimiento de la empresa las experiencias de los clientes en las interacciones con atención al cliente ya que pueden definir el estado de los clientes: promotor o detractor.

En particular, y sustentando esta visión, la compañía Fintech sobre la cual se concentrará este trabajo, tiene planteados desde el año 2022 tres objetivos estratégicos: uno de estos tres objetivos es, justamente, la experiencia de los clientes (medida como NPS)[2].

1.2 Problema

El problema que intentará resolverse en este trabajo es el de predecir el puntaje que pondrán los clientes al finalizar un chat de atención con un agente vía WhatsApp, ante el envío de la encuesta de satisfacción. Ésta tiene la siguiente forma:

"En una escala del 1 (muy mala 👎) al 5 (muy buena 👍), ¿cómo fue tu experiencia con el asesor que te atendió?"

La predicción potencialmente podría permitir la implementación de alguna heurística o herramienta por parte de la empresa para la detección "en vivo" de posibles detractores, con la consecuente posibilidad

[1] Fred Reichheld, La Pregunta Decisiva 2.0, LID Editorial, 1° edición, págs. 74 y 82.
[2] En base a documentos internos de la compañía de presentación anual de objetivos 2022.

de tomar alguna acción en el momento para prevenir la detracción y mejorar la experiencia de los clientes (e.g.: realizar una bonificación, pasar el caso a un supervisor, etc.).

Adicionalmente, no todas las conversaciones entre clientes y agentes tienen una respuesta a la encuesta por dos motivos:

i. No siempre se envía una encuesta al final de una conversación. Se parametrizan reglas para el envío de forma tal de no "saturar" al usuario con encuestas demasiado frecuentes
ii. No siempre los usuarios responden a las encuestas

Según el libro *"La Pregunta Decisiva 2.0"* de Fred Reichheld, quienes no responden las encuestas (ítem ii. más arriba) hubiesen respondido la encuesta con una nota peor que quienes sí lo hicieron. Como se verá más adelante sólo el 36% de las encuestas enviadas tienen una respuesta que permita asignar un puntaje. De esta forma, podría haber una sobreestimación del puntaje del servicio de atención al cliente al no contar con la información de la percepción del servicio de aquellos usuarios que deciden no responder la encuesta.

Por ello, resultaría de interés para la empresa la predicción de puntajes sobre conversaciones que no tuvieron encuesta, de forma tal de contar con mayor información sobre la experiencia brindada.

1.3 Objetivo

En este trabajo se buscará formular un modelo predictivo que permita:

i. Predecir el puntaje con el que un cliente puntuará la atención recibida por un agente a partir de distintas características del cliente, el agente y la conversación
ii. Comprender la importancia para la predicción de las distintas variables disponibles y generadas a partir de las bases disponibles

Para lograr estos dos objetivos se planteará una metodología de trabajo que buscará encontrar *features* que caractericen la interacción entre agente y cliente para intentar predecir el puntaje final que pondrá el cliente. La forma en que se realizará se describe en las próximas secciones. A continuación, se explican las fuentes de datos disponibles para lograr estos objetivos.

2. Datos

Se cuenta con dos grandes fuentes de datos. La primera es la transcripción de los chats vía WhatsApp del servicio de Atención al Cliente de la empresa que cuenta con algunos metadatos que se detallarán más adelante. La segunda, es la información del legajo digital de los clientes de la empresa, la cuál es levantada a partir del registro que hacen de forma autogestionada los clientes al dar de alta su cuenta en la empresa.

Por cuestiones de confidencialidad, las bases no se podrán dejar visibles en este trabajo ya que se trata de información privada de los usuarios. En el caso de precisar mostrar la información de los datos durante el trabajo, ésta podría mostrarse de forma codificada y/o parcial de forma tal de evitar mostrar datos confidenciales, sensibles o que permitan identificar a alguna persona.

En cuanto al origen de las bases es importante destacar que vienen de dos proveedores distintos: la base de clientes viene directamente del sistema CRM de la empresa, mientras que la base de mensajes viene de los servicios de un proveedor de la empresa, el cual provee la plataforma para la comunicación entre agentes y clientes. El principal problema de esto tiene que ver con la "conexión" de los datos entre ambas bases, teniendo un único campo único para conectarlas que, como se verá más adelante, no tiene un *match* perfecto entre ambas bases, lo que implica cierta pérdida de información al unir los datos.

De todas formas, se cuenta con un tamaño de bases iniciales lo suficientemente grande como para que el tamaño final sea relevante y suficiente para el entrenamiento de modelos predictivos.

Otro de los grandes desafíos que presentan los datos es la transformación de los mismos, en particular de la base de mensajes, ya que gran cantidad de la *metadata* que interesa a los fines de este análisis se encuentra dentro de un único campo (*mensaje*) en formato *string*, lo que hace que sea muy relevante la correcta y prolija transformación de ese campo a variables manejables con sus correspondientes *tests* de calidad de datos.

2.1 Marco de trabajo

Como la parte central de los datos proviene de la base de datos de mensajes, vale la pena hacer un breve apartado comentando cómo llega a darse la interacción que genera la información de esta base.

La empresa en cuestión, para la ventana de tiempo estudiada, brinda servicios financieros y del estilo de "billetera virtual" por medio de una aplicación para celulares. Ante consultas, pedidos o reclamos deja a disposición del cliente 3 canales de atención: mail, teléfono y WhatsApp. En el período estudiado, aproximadamente 3 de cada 4 contactos recibidos por la empresa se dan por WhatsApp, resultando este el canal más representativo de la experiencia con el servicio de Atención al Cliente.

El WhatsApp siempre "recibe" al cliente con un *chatbot* de atención. Este *bot* interactúa con los clientes ofreciéndoles menús con opciones o también interpretando palabras libres que dice el cliente. Ante distintas situaciones que pueden darse por las elecciones que hacen los clientes en el menú o por las palabras que le dicen al *bot*, el *bot* puede derivar a los clientes con un agente para que pueda atender su consulta, pedido o reclamo.

A continuación, se detallan algunas definiciones que se utilizarán a lo largo del trabajo.

2.1.1 Definiciones

A los fines de simplificar los conceptos que se utilizan a lo largo de este trabajo, se definen algunos de los principales conceptos que se utilizarán:

Emisor: es una variable categórica que define qué parte envió un mensaje, pudiendo ser: agente, cliente o *bot*.

Agente: parte interviniente en un chat de atención al cliente que brinda atención por parte de la empresa ante consultas de clientes por medio de WhatsApp.

Cliente: parte interviniente en un chat de atención al cliente que busca atención por parte de la empresa por medio de WhatsApp.

Bot: parte interviniente en un chat de atención al cliente que envía mensajes en formato de texto (respuestas) a los clientes ante la recepción de un estímulo (disparador) que puede ser o bien un mensaje enviado por el cliente o una opción seleccionada (botón) por un cliente desde un menú de opciones en la conversación de WhatsApp.

Mensaje: cada uno de los textos enviados en un chat de WhatsApp, equivalente al contenido de cada "globo" de WhatsApp. Los clientes pueden enviar mensajes de audio, que son traducidos automáticamente a texto para la lectura de los agentes y del *bot*. Los agentes y el *bot* no pueden enviar mensajes de audio.

Bloque: grupo de mensajes enviados consecutivamente por un mismo emisor.

Conversación: grupo de mensajes con interacción de agentes y clientes delimitado por eventos de inicio y eventos de fin.

Evento de inicio: mensajes que dan inicio a una conversación, marcados por un evento de derivación a un agente (e.g..: cuando a partir de un mensaje del cliente, el *bot* responde con un mensaje que indica "Te estoy derivando con un asesor").

Evento de fin: mensajes o sucesos que marcan el final de una conversación, definidos por las siguientes 3 condiciones:

- "*No more wait*": cuando un cliente es derivado a un asesor y no hay uno disponible inmediatamente, el *bot* ofrece al cliente dos opciones: continuar esperando o salirse de la cola de espera. Si el cliente elige salirse de la cola de espera, se considera un evento de fin del tipo *no_more_wait*.
- "*Day end*": el servicio de atención al cliente de la empresa termina a la medianoche, por lo que cada cambio de día calendario se considera un evento de fin del tipo *day_end*.
- "*Agent close*": cuando los agentes desde su consola de atención hacen clic en el botón de cerrar conversación, dejan de tener la posibilidad de intervenir en la conversación, y vuelve a intervenir el *bot* enviando un mensaje con la encuesta para evaluación del agente. Esto se considera un evento de fin del tipo *agent_close*.

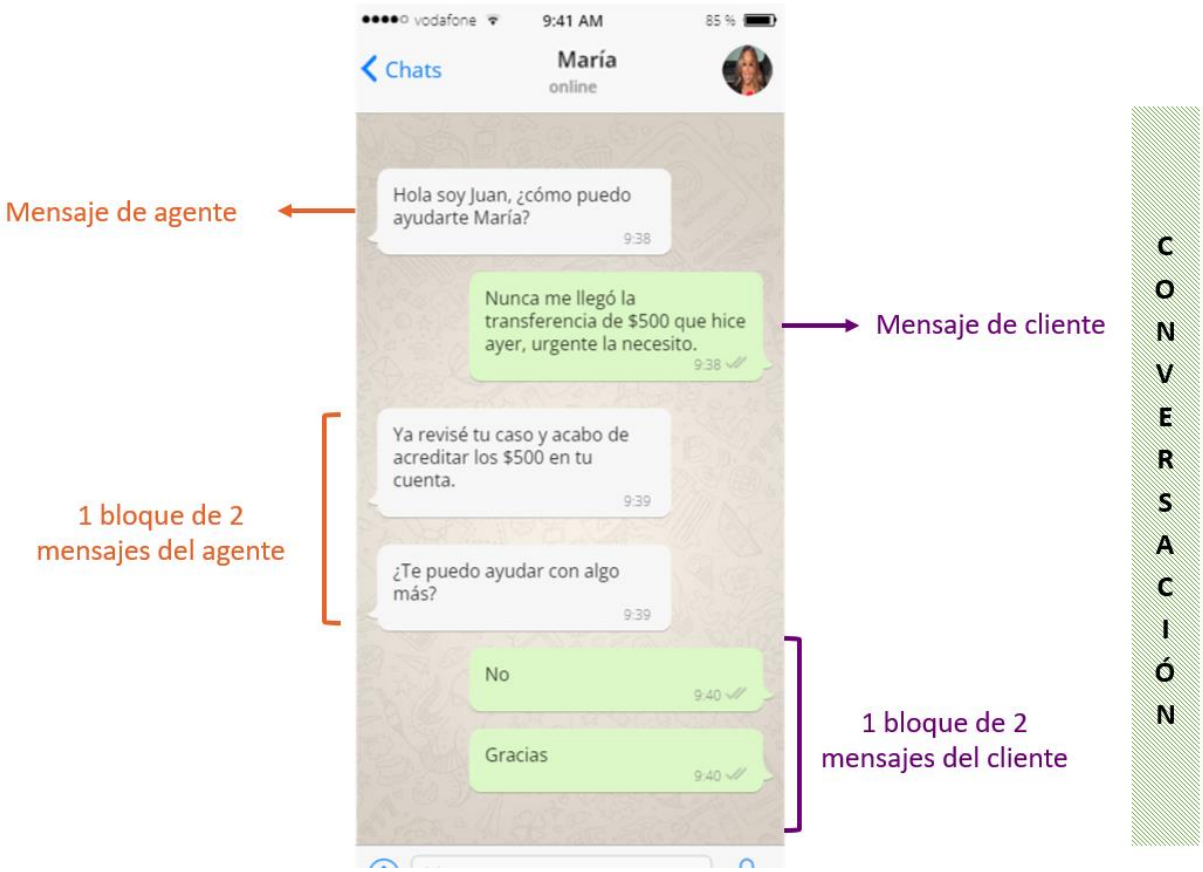
Encuesta: cuando un agente hace clic en el botón de cerrar conversación, se envía un mensaje al cliente el cual contiene la siguiente pregunta: "En una escala del 1 (muy mala 👎) al 5 (muy buena 👍), ¿cómo fue tu experiencia con el asesor que te atendió?". Según algunos parámetros definidos por el equipo de atención al cliente, existen algunas excepciones en las que no se envía la encuesta al cerrar la conversación, por ejemplo: cuando se envió una encuesta hace muy poco tiempo a ese mismo cliente. La respuesta del cliente a la encuesta, es lo que define la variable dependiente y se llama "Puntaje".

Puntaje: el puntaje considerado para las encuestas toma en cuenta los números del 1 al 5, siendo el resto de los valores considerados como no válidos. Se considera *Promotor* a todo puntaje de 5. Se considera *Detractor* a todo puntaje entre 1 y 4 inclusive. Este puntaje, será la variable a predecir del modelo.

2.1.2 Estructura de un chat

Para facilitar el entendimiento de la dinámica de las conversaciones, a continuación, se muestra un ejemplo de un chat, indicando los distintos elementos que lo componen:

Figura 1: pantallas mostrando la estructura de un chat y sus componentes


vodafone 9:41 AM 85 %
Chats
María
online
Hola soy Juan, ¿cómo puedo ayudarte María?
9:38
Nunca me llegó la transferencia de $500 que hice ayer, urgente la necesito.
9:38
Ya revisé tu caso y acabo de acreditar los $500 en tu cuenta.
9:39
¿Te puedo ayudar con algo más?
9:39
No
9:40
Gracias
9:40
Mensaje de agente
Mensaje de cliente
1 bloque de 2 mensajes del agente
1 bloque de 2 mensajes del cliente
CONVERSACIÓN


Movistar 9:41 AM 85 %
Chats
María
online
Te dejo una breve encuesta:
9:40
En una escala del 1 (muy mala 👎) al 5 (muy buena 👍), ¿cómo fue tu experiencia con el asesor que te atendió?
9:40
5
9:41
¿Nos querés contar los motivos de tu evaluación?
9:41
Resolvió rápido
9:41
Gracias por escribirnos, sigo acá para lo que necesites. ¡Un abrazo! 😊
9:41
Mensaje del bot
Mensaje del bot (envío de encuesta)
Mensaje del bot (pregunta motivos)
Mensaje del bot
Evento de final de conversación (cierre de agente)
Mensaje del cliente (puntaje encuesta)
Mensaje del cliente (motivos encuesta)
CONVERSACIÓN

2.1.3 Estructura de datos

A continuación, se presenta un esquema resumido de la representación de los datos de la conversación anterior, similar a cómo se observaría en el *dataframe train_data* presente en *script_mensajes.py*. Cada línea se corresponde a un mensaje, según fuera definido en la sección de definiciones de la página 9.

Para mayor claridad, se destacan en celeste los mensajes enviados por el *bot*, en violeta los mensajes enviados por el cliente y en naranja los mensajes enviados por el cliente:

Tabla 1: estructura resumida de datos de una conversación

mensaje	*conv_event*	*conv_id*	*conv_seq*	*block_id*	*score_agent*	*score_agent_conv_id*
no veo mi plata				223		
¿Qué información te interesa? Elegí una de las siguientes opciones ??				224		
¡Hola María! ?? Soy el asistente virtual y estoy para ayudarte con tus consultas				224		
A. Cuenta B. Recargas C. Cobros D. Pago de resumen E. Otros				224		
Ninguna, necesito hablar con un agente				225		
Te estoy derivando con un asesor	derivation	77	1	226		
Hola soy Juan, ¿cómo puedo ayudarte María?		77	2	227		
Nunca me llegó la transferencia de $500 que hice ayer, urgente la necesito		77	3	228		
Ya revisé tu caso y acabo de acreditar los $500 en tu cuenta		77	4	229		
¿Te puedo ayudar con algo más?		77	5	229		
No		77	6	230		
Gracias		77	7	230		
Te dejo una breve encuesta:	agent_close	77	8	231		
En una escala del 1 (muy mala ??) al 5 (muy buena ??), ¿cómo fue tu experiencia con el asesor que te atendió?				231		
5				232	5	77

mensaje	*conv_event*	*conv_id*	*conv_seq*	*block_id*	*score_agent*	*score_agent_conv_id*
¿Nos querés contar los motivos de tu evaluación				233		
Resolvió rápido				234		
Gracias por escribirnos, sigo acá para lo que necesites. ¡Un abrazo! ??				235		

Algunos puntos importantes para destacar sobre la tabla anterior:

1. Los mensajes por sí mismos podrían tener alguna información predictiva para la variable dependiente, pero se percibe en este primer vistazo de los datos que hay una gran oportunidad en mejorar el valor informativo de esta estructura de datos, pasando de una estructura de mensajes individuales a conversaciones que agrupen mensajes y capturen interacciones.

2. De las conversaciones surgirán varias métricas que llamaremos "conversacionales", las cuales podrán traducir en información de valor algunas características que se ven en la Tabla 1. Por ejemplo: la cantidad de mensajes seguidos enviados por un mismo emisor podría indicar características de insistencia o impaciencia de los emisores.

3. Siendo que cada mensaje es una línea, la cantidad de observaciones obtenidas en el período de ~7 meses estudiado es muy grande, y ronda los 14 millones de observaciones.

4. El orden de los bloques dentro de los datos refleja lo sucedido en la conversación real de WhatsApp. No sucede lo mismo con el orden de los mensajes dentro de los bloques, como puede observarse por ejemplo en el bloque 224. En este caso, el orden de los mensajes es correcto sólo cuando nos encontramos en mensajes enviados por agentes o clientes. Pero cuando se trata de mensajes enviados por el *bot*, esto no es necesariamente así.

Esto sucede, porque dentro de la plataforma que ejecuta el envío de mensajes automáticos del *bot*, hay veces en las que, ante ciertos estímulos, se desencadena una respuesta que podría contener el envío de 3 mensajes consecutivos. Por ejemplo, en la Tabla 1, ante el mensaje del cliente "*no veo mi plata*" (estímulo), el *bot* responde con una repuesta compuesta por 3 mensajes. Como puede verse en la Figura 1, el orden de estos mensajes en la tabla de datos no siempre se corresponde con lo sucedido en la conversación. El motivo de esto tiene que ver con que la tabla de origen de datos contiene un *timestamp* del momento del envío de los mensajes con una precisión de ± 1 segundo. Como los mensajes consecutivos enviados por el *bot*, tiene diferencias de milisegundos, estas no son capturadas por la tabla de datos que contiene el *timestamp* truncado al segundo más cercano, resultando en este desorden de los datos.

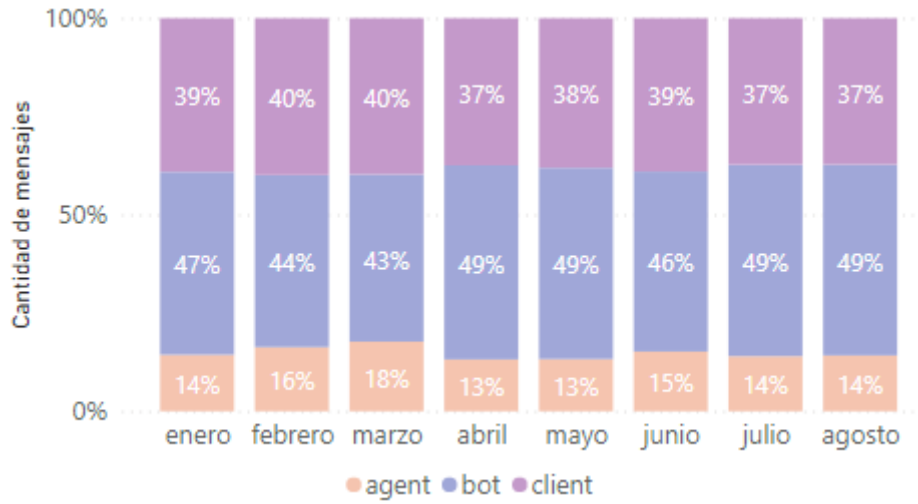
Análisis exploratorio de los datos

A continuación, se muestra la exploración inicial realizada de los datos de estudio, analizando únicamente un set de datos separados como datos de entrenamiento (70% de los datos totales, es decir unos ~10 millones de mensajes). La metodología de separación de estos datos se describirá más adelante en la sección Metodología de la página 19, junto con otras de las transformaciones realizadas sobre los datos iniciales que darán luz sobre nuevas métricas y su distribución sobre los datos.

Los datos corresponden a un *dataset* de una compañía financiera y describen la interacción del canal de WhatsApp de atención al cliente de esa compañía. Corresponden al período que va desde el 1/1/2020 hasta el 5/8/2020.

Inicialmente se busca explorar la proporción de mensajes enviados por cada emisor:

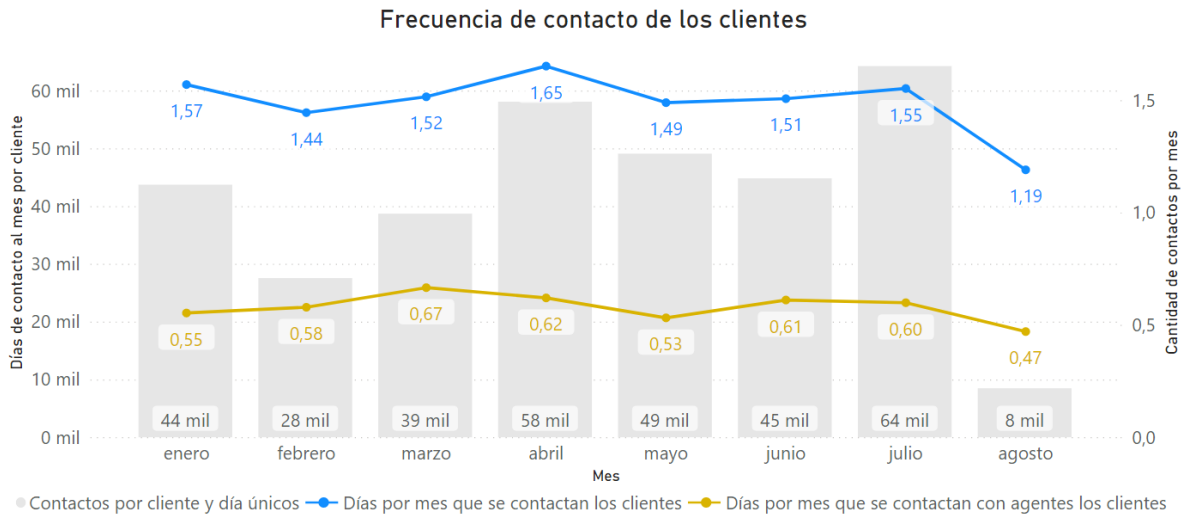
Figura 2: evolución de la proporción de mensajes por emisor



Como puede observarse, cerca de la mitad de los mensajes son enviados por el *bot* en forma de mensajes precargados ante distintos estímulos de los clientes. Si bien esta porción de los datos puede contener información predictiva sobre la puntuación otorgada por los clientes a los agentes, se espera que gran parte de la información que nos permita predecir se encuentre en la otra mitad de los datos: los mensajes enviados por agente y cliente. Puede observarse que hay una relativa estabilidad en el *mix* de estos datos a lo largo del tiempo, sugiriendo una continuidad temporal en la disponibilidad de información de agentes y clientes a lo largo del *dataset*.

En lo que respecta a la frecuencia de contacto de los clientes, podemos observar algunos datos interesantes en el siguiente gráfico:

Figura 3: evolución de la frecuencia de contacto de clientes



Las barras grises describen la cantidad de contactos por cliente y día únicos. Es decir, si el cliente A se contactó 5 veces el tercer día del mes y el cliente B se contactó 2 veces el tercer día del mes y una vez el décimo día del mes, se contabilizarán 3 contactos en esta métrica correspondientes a:

- ➔ cliente A tercer día del mes
- ➔ cliente B tercer día del mes
- ➔ cliente B décimo día del mes

Puede observarse variabilidad en la cantidad de contactos por cliente y día únicos. Esto tiene que ver con motivos del negocio y la operación, ya que en los meses que se muestra mayor cantidad de chats, esta cantidad está relacionada con contingencias (errores masivos en la aplicación) de similar proporción en volumen de usuarios afectados. Por ejemplo, contingencias de demora de impacto de transferencias en cuentas destino durante el mes de abril, las cuales hacen que aumente la tasa de contacto con atención al cliente. Cabe recordar que los datos de agosto no son representativos al contar con datos sólo hasta el 5 de agosto.

En cuanto a la cantidad y calidad del tipo de interacciones, vemos que los clientes que se contactan lo hacen en promedio 1,53 días por mes (línea azul). Asimismo, si contabilizamos los días que se contactan y que además son transferidos a un agente (i.e.: no tienen sólo interacción con el *bot*, sino con el *bot* y con un agente), esta métrica se reduce a 0,59 días por mes (línea amarilla). De estas métricas se concluye que el 61% de los días que se contactan lo hacen sólo interactuando con un *bot*, métrica que se conoce en la industria como *human avoidance* o *human deflection*. El restante 39% tienen interacciones con agentes; en estas últimas interacciones es donde se genera nuestra variable de interés, es decir: la puntuación de la atención del agente.

Al ver cómo es la dinámica de las conversaciones, surgen algunos datos interesantes en la exploración de los datos:

Figura 4: características principales de las conversaciones por emisor

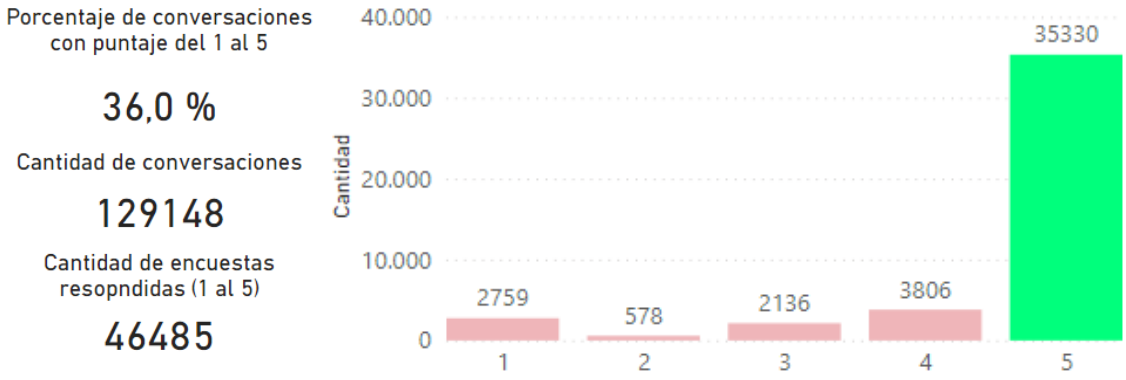
En primer lugar, puede observarse la cantidad de mensajes consecutivos, o mensajes por bloque, que envía en promedio cada tipo de emisor, siendo el *bot* el de mayor cantidad de mensajes por bloque (2,21), seguido por los agentes (1,74) y luego los clientes (1,37). Observamos entonces que los agentes envían 27% más mensajes por bloque que los clientes.

Cuando se ve la frecuencia de mensajes por conversación (recordar definición de conversación en la página 10), puede observarse un comportamiento inverso: en este caso los agentes envían menos mensajes totales por conversación (12,9) que los clientes (16,3). En promedio un 21% menos mensajes por conversación.

Estas dos conclusiones parecieran ser contradictorias ya que, a lo largo de una conversación entre agente y cliente, se esperaría que haya de forma alternada 1 bloque por cada emisor, por lo que el emisor que tenga mayor tasa de mensajes por bloque debería tener la mayor cantidad de mensajes por conversación. Esto no resulta ser así debido a que, tal como está definida la conversación, se generan mensajes de los clientes de largo de bloque = 1 durante algunas interacciones **dentro de la conversación** en las que interactúa **el *bot***. En este caso, al ser derivados a un agente, muchas veces se genera una intervención del *bot* cuando hay una espera elevada en la atención preguntando a los clientes si desean continuar esperando. Ante esto, se genera una respuesta de los clientes generalmente de 1 mensaje (largo de bloque = 1) que eleva la métrica de mensajes por conversación y reduce el promedio de mensajes por bloque para el cliente.

En lo que respecta a la exploración de la variable dependiente, a continuación se ve la información sobre el puntaje:

Figura 5: distribución del puntaje otorgado por los clientes a los agentes



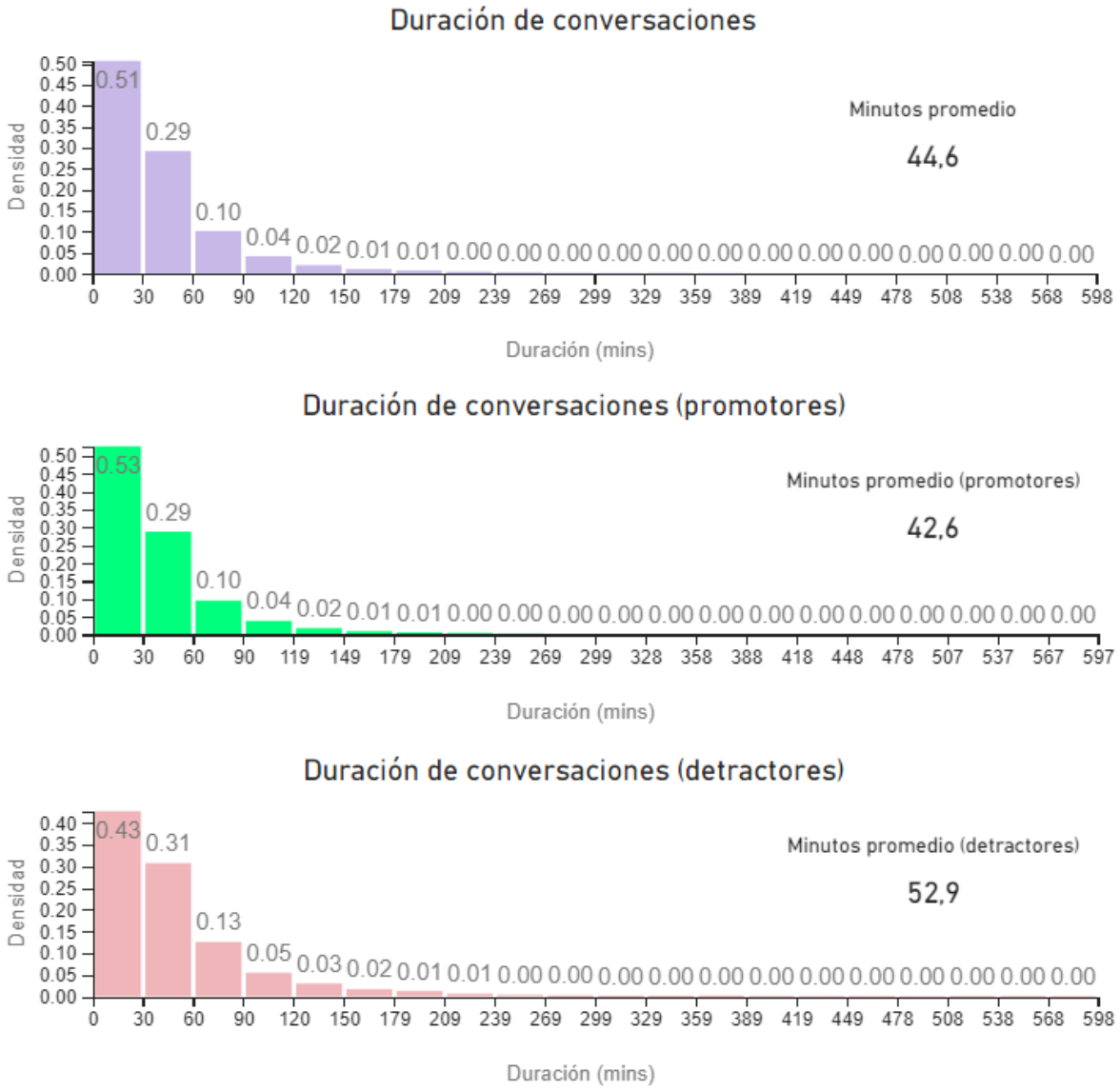
Como primer punto, es importante destacar que no todas las conversaciones tienen un puntaje valido del 1 al 5: sólo el 36% lo tiene. Al ser una conversación de WhatsApp, la nota de los clientes no tiene una validación de datos, sino que es un texto libre a ingresar por el cliente en su WhatsApp. Por eso nos encontramos con muchas encuestas no respondidas o respondidas con texto o valores mal escritos.

En segundo lugar, se tiene ahora por primera vez en el trabajo noción de la cantidad de "observaciones reales" que tenemos. En este set de datos de entrenamiento nuestras "observaciones" ya no son los ~10 millones de mensajes registrados, sino estas 46.485 encuestas respondidas del 1 al 5, con sus correspondientes *features* a construir a partir de las conversaciones que derivaron en las encuestas.

Por último, vemos que la distribución de los puntajes tiene una tendencia hacia el puntaje más alto, concentrando el 79% de las encuestas en el puntaje 5. Si bien no parece ser un desbalance fuerte, es un punto a tener en cuenta para tener ciertos cuidados a la hora de evaluar los modelos predictivos.

Para concluir con el análisis exploratorio de datos veremos una última característica de las conversaciones, su duración:

Figura 6: distribución de la duración de las conversaciones



En los gráficos podemos ver la distribución de la duración de las conversaciones desde un evento de inicio hasta un evento de fin, agrupada por 3 categorías: conversaciones totales, conversaciones con puntaje 5 (promotor) y conversaciones con puntaje del 1 al 4 (detractor).

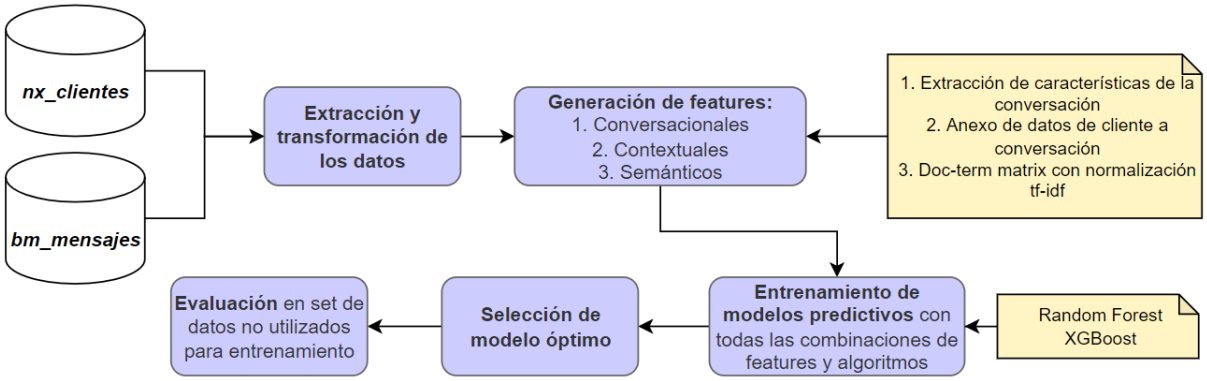
Se observa que en promedio las conversaciones de los detractores tienen una duración 24% mayor (52,9 minutos) en comparación a los promotores. Esto nos acerca una sugerencia que podríamos validar en el modelo predictivo respecto al valor explicativo de las variables de tiempo dentro de la conversación. Arriesgando una interpretación temprana, las conversaciones más largas podrían estar reflejando conversaciones más complejas, con mayor demora en poder encontrar una resolución, lo que podría explicar una tendencia a puntajes más bajos por la menor velocidad en resolución/conclusión de la conversación.

3. Metodología

Todos los procesos descriptos en esta sección, con excepción del último paso (Evaluación en datos de *test*, en la página 48), se realiza sobre los datos de entrenamiento o *train*, es decir, sobre un conjunto del 70% de los datos originales. El proceso de separación de este set de entrenamiento, del set de *test* o evaluación, se describe más adelante.

A continuación, se muestra el esquema general de alto nivel que describe la metodología de trabajo que se utilizará en este trabajo:

Figura 7: esquema de trabajo para creación y evaluación del modelo predictivo



El esquema de trabajo cuenta con 5 grandes pasos que se describen a continuación, y se profundizarán en las secciones siguientes:

- **Extracción y transformación de los datos.** Carga de datos desde origen y remanejo para la correcta adaptación al entorno para la generación de *features*, para la interpretación de los datos "sucios" y para poder ser utilizados como *input* para los modelos predictivos.
- **Generación de *features*.** Creación de nuevas variables, adaptación de variables existentes y *join* de los *features* de diversos orígenes para la generación de *features* con potencial explicativo de la variable dependiente.
- **Entrenamiento de modelos predictivos.** Refinamiento de hiperparámetros de los algoritmos y *features* seleccionados en los datos de entrenamiento.
- **Selección de modelo óptimo.** Selección de mejor combinación de algoritmos y *features* según los resultados de entrenamiento.
- **Evaluación.** Aplicación del modelo predictivo sobre datos *held-out* para evaluación de *performance* predictiva de éste.

3.1 Origen de datos

Para este trabajo se utilizaron 2 bases de datos de una empresa Fintech argentina.

La primera base de datos es la base *bm_mensajes*, que contiene la información de los mensajes enviados y recibidos en el principal punto de contacto para la atención de la empresa, es decir, el canal de WhatsApp. La base inicial sobre la cual se trabajo fue descargada mediante sucesivas consultas direccionadas al *datalake* de la empresa para poder descargar la información en partes, cada una correspondiente a un rango de entre 9 y 45 días de mensajes. La información fue descargada a 16 archivos *.csv* que luego fueron utilizadas desde el código en *script_mensajes.py*.

La segunda base de datos es la base *nx_clientes*, que contiene la información correspondiente a los datos filiatorios de los clientes de la empresa al momento de registro en la aplicación. Cabe destacar, que los datos requeridos al momento del registro fueron variando a lo largo del período de tiempo que cubre esta base, además de existir algunos datos opcionales. Por este motivo, es de esperar que no todas las columnas tengan los datos completos para todas las observaciones (clientes).

A continuación, se resume la información principal de las dos fuentes de datos:

Tabla 2: características principales de las bases de datos utilizadas

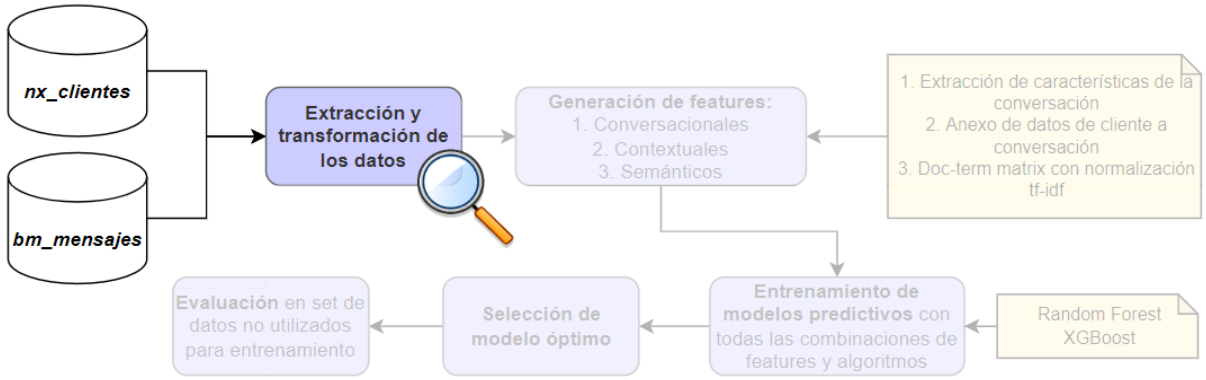
Base de datos	Dimensiones [filas x columnas]	Tipo de datos en filas	Tipo de datos en columnas	Rango temporal de datos
bm_mensajes	14.069.242 x 15	Mensajes enviados o recibidos	Emisor del mensaje, fecha del mensaje, contenido del mensaje, teléfono del cliente, etc.	1/1/2021 al 5/8/2021 (fecha envío mensaje)
nx_clientes	2.241.530 x 87	Clientes registrados en la app	Id interno del cliente, nombre, apellido, sexo, estado civil, teléfono del cliente, fecha de registro, etc.	9/9/2019 al 30/10/2021 (fecha registro cliente)

El detalle de las columnas más relevantes de cada base de datos puede encontrarse en el Apéndice A: Descripción de los datos de origen

.

3.2 Extracción y transformación de los datos

En esta sección se detallará el proceso de extracción y transformación de los datos:



Este proceso se realizó en 5 etapas que se explicarán a continuación, siguiendo la línea de las secciones 1 a 5 del script *mensajes.py*.

Sección 1 – Preparación y carga de *datasets*

En la sección 1 del código se realiza la carga de los archivos de la base de datos *bm_mensajes* desde el directorio de trabajo. La carga de la base *nx_clientes* se realiza de forma similar sólo que, por comodidad para el seguimiento del código, se realiza al momento de la etapa de generación de *features* semánticos.

A los fines de minimizar el riesgo de *data leakage*, es en este primer paso que se realiza la separación de lo que llamaremos los datos de *train* (70% de la base *bm*_mensajes) y los datos de *test* (30%). En el resto de este script no se utilizarán más los datos de *test* y se dejarán de lado hasta llegar a la etapa de evaluación.

En la función `train_test_subset` del código se hace la asignación de cada observación de la base de datos de mensajes al grupo *train* o al grupo *test*. Se realiza con una asignación de cada *id_contacto* único (ver Apéndice A: Descripción de los datos de origen

para detalles del campo) de manear aleatoria con una probabilidad del 70% de asignación de la etiqueta *train* y 30% de la etiqueta *test*.

Finalmente, utilizando la función `train_test_split` se separa la base de datos cargada en 2, según las etiquetas asignadas a cada *id_contacto*.

Para seleccionar la lógica de separación en *train* y *test*, se evaluaron distintos criterios:

- **Criterio de mensajes aleatorios**: los mensajes se mezclan de forma aleatoria y se distribuyen en grupos de *train* y *test*. Se descartó esta opción ya que se pierde la estructura de conversación definida anteriormente, y se tendrían sólo mensajes sueltos, lo que haría que se pierda mucha información con potencial de predicción.
- **Criterio de temporalidad de mensajes**: los mensajes se mantienen en el orden cronológico en el que se encuentran, sin perder la estructura de conversación. Se deja en *train* el 70% de las conversaciones más antiguas, ordenadas cronológicamente y en *test* el 30% más reciente. Se descartó esta opción por dos motivos principales: a) podría existir un sesgo temporal por estacionalidad o eventos puntuales de algunos meses (e.g.: contingencias en el funcionamiento de la aplicación en un mes dado que afecten el puntaje otorgado a los agentes de atención al cliente) y b) al intentar predecir, en la etapa de evaluación, el comportamiento de clientes de *test*, lo estaríamos haciendo con un modelo que ya "vio" información de esos clientes en el entrenamiento, lo cual generaría un *data leakage*.
- **Criterio de clientes**: la separación en *train* y *test* se realiza separando por completo los clientes en un grupo y otro, de forma tal que no existan clientes que registren interacciones tanto en *train* como en *test*. En otras palabras: cada cliente puede estar sólo en uno de los dos grupos. Este es el criterio que se utilizó.

Al momento de pensar en este criterio vs. el criterio de mensajes aleatorios, es cuando surge a la vista la importancia de la definición de una estructura de conversación, explicada más atrás, y sus partes, ya que, dada esta estructura, pueden extraerse muchos *features* que *a priori* y de forma intuitiva parecieran ser candidatos a tener cierto poder predictivo sobre la variable dependiente, como, por ejemplo: la demora en la respuesta de los agentes, la duración de la conversación, etc.

La lógica detrás de la separación de los conjuntos de datos de *train* y *test* a partir del teléfono de contacto (campo *id_contacto*) radica en que este campo es el campo que será la clave primaria para unir los mensajes con el resto de los datos contextuales de los clientes, disponibles desde la base de datos *nx_clientes*. De esta forma, cualquier información que anexemos a la información de esta base de mensajes, por definición también estará separada correctamente en *train* y *test* evitando el *data leakage*.

Sección 2 – *Sorting* y *tagging*

En la sección 2 se comienza realizando el ordenamiento de los mensajes mediante la función `sort_data`. Esta función ordena los mensajes por los campos *id_contacto* y *fecha_creacion*. De esta forma se tiene una secuencia de mensajes que ordena a los clientes por número telefónico y dentro de cada grupo de mensajes asociados a ese cliente (sean mensajes con emisor *bot*, agente o cliente), los

ordena por la fecha en que fueron enviados. Así, tenemos la historia de interacciones de cada cliente ordenada cronológicamente.

En lo referente al *tagging*, en esta sección se comienzan a identificar algunos elementos de las conversaciones que serán los ladrillos que construirán la macroestructura de conversación ya definida y algunos otros eventos que permitirán construir algunos *features* más adelante. De esta forma se incorporan al *dataframe* en forma de nuevas columnas, la detección de ciertos eventos como ser:

- ➔ La selección de la opción "No" por parte del cliente, ante ciertos menús de opciones que ofrece el *bot* (e.g.: ¿Te gustaría seguir esperando? > "No")
- ➔ La selección de la opción "Sí" por parte del cliente, ante ciertos menús de opciones que ofrece el *bot* (e.g.: ¿Te gustaría seguir esperando? > "Sí")
- ➔ La detección del evento *wait_offer*, definido como el envío de parte del *bot* del mensaje de "¿Te gustaría seguir esperando?" a un cliente que pidió hablar con un agente y aún no fue atendido.
- ➔ La detección (de una forma más amigable que como viene en los datos de origen) de que el emisor de un mensaje fue un agente.

Sección 3 – Eventos de inicio y de fin de conversación

En esta sección se delimita en el *dataset* la estructura de conversación definida en la sección de Definiciones de la página 9. De esta forma, se marcan y anexan como columnas el evento de inicio (derivación a asesor) y los eventos de fin de una conversación (final del día, respuesta del cliente de no querer seguir en espera y cierre de conversación por parte del agente).

Estas columnas dejan estructurados los datos para la siguiente sección.

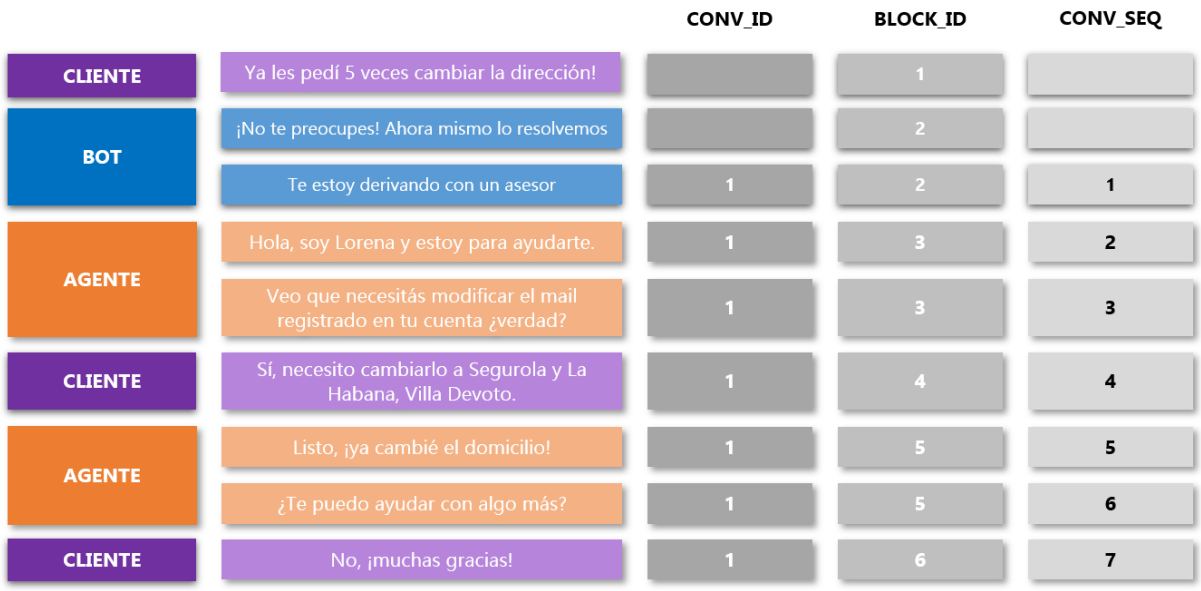
Sección 4 – Secuenciación

Una vez definido el inicio y final de las conversaciones, cobra relevancia entender la secuencia en la que se dan los mensajes dentro de la conversación. Para eso se realizan tres acciones, principalmente, en las siguientes funciones:

- ➔ `assign_conv_id`: se asigna un identificador numérico y único a cada conversación para poder contabilizarlas e identificarlas.
- ➔ `sequence_conversations`: se asigna un identificador único dentro de la conversación a cada mensaje, que identifica el ordinal del mensaje dentro de la conversación.
- ➔ `assign_block_id`: para cada grupo o bloque de mensajes consecutivos enviados por el mismo emisor (*bot*, agente o cliente) se asigna un identificador único dentro de la conversación que identifica a ese bloque de mensajes con un *block_id*.

De esta forma se consigue la noción de orden secuencial a tres niveles: conversación, bloque y mensaje.

Figura 8: representación de una conversación de ejemplo son su etiquetado de conv_id, block_id *y* conv_seq



Sección 5 – Variable objetivo

La variable a predecir no se encuentra en los datos crudos separada en una columna aparte, sino que se encuentra como un texto enviado por los clientes posterior a la pregunta del *bot*: "En una escala del 1 (muy mala 👎) al 5 (muy buena 👍), ¿cómo fue tu experiencia con el asesor que te atendió?". Por ese motivo, es relevante "limpiar" la información, ya que en los datos nos encontramos con respuestas a la pregunta de este estilo:

- ➔ "0"
- ➔ "Excelente"
- ➔ "Todavía no resolvieron mi consulta"
- ➔ "10"
- ➔ Y otros casos que no tienen respuesta.

Además de este problema, nos encontramos con el problema de la secuenciación de las observaciones: capturar la respuesta no resulta tan simple como observar la observación siguiente a la línea que contiene la pregunta de la satisfacción del cliente, ya que podrían suceder variaciones del orden de los mensajes como los que se ven en este fragmento de los datos reales utilizados:

Tabla 3: ejemplo de envío de encuesta con orden de mensajes distinto al real

Emisor	Mensaje
BOT	En una escala del 1 (muy difícil ??) al 5 (muy fácil ??), ¿qué tan fácil fue gestionar tu consulta por WhatsApp?
BOT	Te dejo una breve encuesta:
CLIENTE	Gracias
BOT	Por favor, ingresá un valor del 1 al 5
BOT	¡Que tengas un buen día! ??

Como se observa en este ejemplo, la respuesta del cliente a la encuesta aparece 2 líneas debajo de la encuesta, cuando el cliente dice *"Gracias"*, es decir, en este caso no ingreso un número del 1 al 5 válido. Al detectar una respuesta inválida, el *bot* vuelve a pedir que ingrese un valor válido del 1 al 5. Al no obtener respuesta por determinado tiempo, el *bot* concluye la encuesta indicando *"¡Qué tengas un buen día!"*.

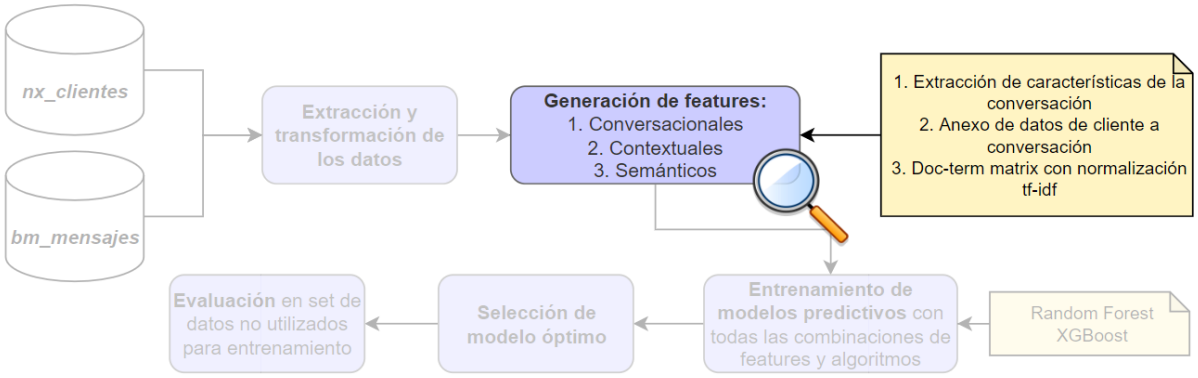
Para capturar la mayor cantidad de respuestas, se implementó una función que pueda detectar en qué línea el cliente da la respuesta, y en caso de que tenga una repuesta invalida, volver a intentar capturar la respuesta en el siguiente intento del *bot*. Además, la función captura únicamente las respuestas que contienen los números del 1 al 5[3].

Por último, para facilitar el manejo de los datos, y la construcción de los *dataframes* finales para los modelos predictivos, a cada respuesta se le anexó la conversación (*conv_id*) de la cual provino esa nota, en el campo *score_agent_conv_id*.

Todo esto fue implementado en la función `extract_agent_score` dentro de *script_mensajes.py*.

3.3 Generación de *features*

En esta sección se detallará el proceso de generación de features:



Este proceso se realizó en 4 etapas que se explicarán a continuación, siguiendo las secciones 6 a 9 de *script_mensajes.py*.

En estas secciones se toman los datos ya pre-procesados de *bm_mensajes* y los datos crudos de *nx_clientes* para generar *features* esperando que tengan poder predictivo sobre la nota de evaluación a los asesores de atención.

Se definen, entonces, 3 categorías de *features*:

- ***Features* contextuales**: se trata tanto de características del usuario como del contexto en que se da el contacto con atención al cliente. Acá se encuentran variables demográficas (e.g.: nacionalidad, código postal, sexo, etc.) como variables que describen la situación o contexto del cliente al momento del contacto con atención al cliente (e.g.: día de la semana en que ocurre el

[3] Podría considerarse que existe cierto valor en las respuestas del tipo "0" (cliente para nada satisfecho) o "10" (cliente muy satisfecho que respondió utilizando una escala más tradicional en la vida cotidiana de los clientes), pero, además de que el porcentaje de estas respuestas es despreciable, se decidió apegarse estrictamente a aquellos valores que son exactamente 1, 2, 3, 4 o 5 para no agregar ninguna interpretación subjetiva a lo escrito por el cliente. Por ejemplo, el "0" podría ser un cliente muy poco satisfecho, pero también podría ser un cliente que escribió mal y quiso poner un "10", entonces: ¿deberíamos asignar esta respuesta a la nota 1 o a la nota 5?

contacto, antigüedad del cliente en la aplicación Fintech al momento del contacto, momento del ciclo de cobranza al momento del contacto[4], etc.)

- ***Features* conversacionales**: se trata de características internas propias de la dinámica de la conversación. A diferencia de los *features* contextuales, los *features* conversacionales no miran en qué situación o contexto llega el cliente al contacto, sino que miran de qué forma sucede el contacto. Aquí se encuentran *features* del estilo de: cantidad de mensajes totales de la conversación, si el cliente tuvo que esperar previo a ser atendido, si el cliente envió imágenes, etc.
- ***Features* semánticos**: estos *features*, como se explicará luego, tienen una forma muy específica y capturan de forma normalizada la frecuencia de ocurrencia relativa de las palabras utilizadas por agentes y por clientes por separado, para cada conversación.

Sección 6 – *Features* contextuales

Para la generación de *features* contextuales, se incorporó la información de la base de datos *nx_clientes*. A partir de esta base y los datos pre-procesados de la base de datos *bm_mensajes* se obtuvieron los *features* contextuales de dos formas:

1. Algunos *features* simplemente fueron extraídos seleccionando las columnas correspondientes en la base *nx_clientes* y utilizándolas como *feature* en sí mismas (e.g.: nacionalidad).
2. Otros *features* fueron generados a partir de cierta lógica derivada del momento en qué se dio el contacto.

A continuación, se describen todos los *features* contextuales extraídos y, en caso de haber tenido alguna lógica para su generación, esta se describe:

[4] La tarjeta de crédito es el producto central de la tarjeta Fintech que se está analizando y el tipo de interacciones que pueden llegar a darse podría variar según el ciclo de cobranza. Esta variable se referirá a en qué momento del ciclo de pago de resumen se encuentra el usuario: entre el cierre y la fecha de vencimiento o entre la fecha de vencimiento y el cierre.

Tabla 4: descripción de los features *contextuales*

Origen	***Feature***	Descripción
nx_clientes	*fecha_creacion*	Fecha de registro del cliente en la aplicación
	nacionalidad	País de origen del cliente
	provincia	Provincia de residencia del cliente
	sexo	Sexo del cliente: masculino o femenino
	persona_fisica	Variable binaria que indica si el cliente es persona física o jurídica
	estado_civil	Descripción del estado civil del cliente, cargado al momento de registro.
	fiscal_position	Descripción de la condición tributaria del cliente: monotributista, consumidor final, etc.
	new_postalcode	Código postal del cliente
bm_mensajes	*tag_weekday*[5]	Día de la semana en que se inició la conversación con el agente
	tag_collections_cycle[6]	Momento del ciclo de tarjeta de crédito en que se encuentra el cliente al momento de la conversación. Toma el valor "cobranza" entre el día 26 y el día 10 del mes siguiente, que es cuando ya se cerró el resumen de la tarjeta y el cliente se encuentra en posibilidad de pagar el resumen y toma el valor "consumo" entre el día 11 y 25 de cada mes, momento en que ya pasó el vencimiento del resumen y el cliente se encuentre "consumiendo" hasta el siguiente cierre de ciclo el día 26.
Ambas bases de datos	*tenure*[7]	Indica la antigüedad del cliente, medida en días transcurridos desde el registro en la aplicación hasta el día del contacto con el agente.

Un tema relevante en este apartado es que se están uniendo dos bases de datos y vale la pena explicar cómo se da esa unión.

Para poder anexar la información contextual proveniente de la base *nx_clientes* a los datos pre-procesados de la base *bm_mensajes*, se utilizó el número de celular como clave primaria para esta unión. Vale la pena aclarar el origen de estos datos en ambas bases:

- ➔ *nx_clientes*: el dato de número de celular proviene del número de teléfono registrado por los clientes al momento de registro en la aplicación.
- ➔ *bm_mensajes*: el dato de número de celular proviene de la captura automática del celular desde el cual se está contactando el cliente.

Entonces, es importante tener en cuenta ciertas aclaraciones y las verificaciones realizadas para asegurarse que no generarán una distorsión significativa en los datos:

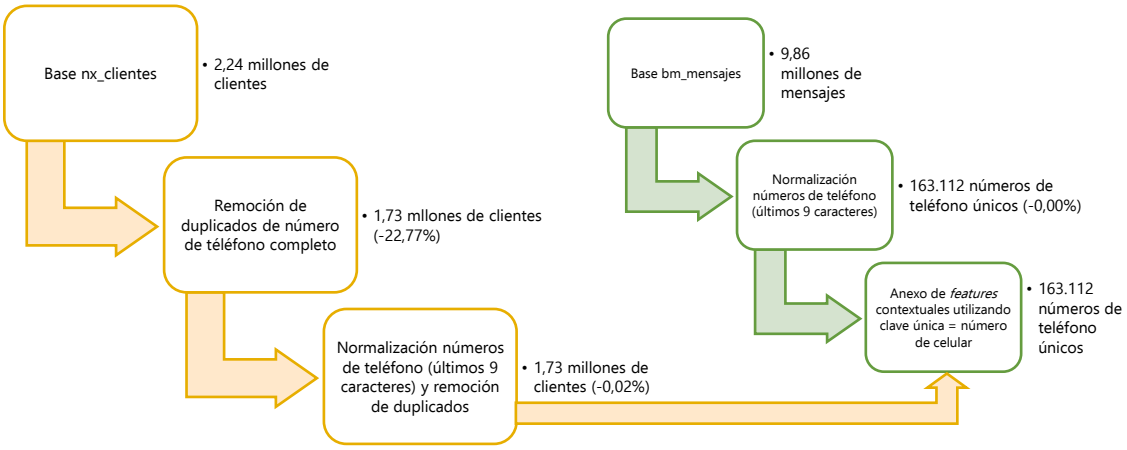
a) *El cliente podría haber escrito un número de celular falso al momento del registro, entonces podría asociarse una conversación con datos contextuales erróneos*. Si bien esto es verdad, las probabilidades son muy bajas ya que, en el escenario de mínima existen al menos 24 millones de combinaciones posibles de códigos de área y teléfono que puede ingresar un usuario, por lo que las probabilidades de que ingrese un número que no es propio y este número coincida con el de otro cliente son extremadamente bajas[8]. El riesgo real en este caso es la pérdida de información, ya que, si un usuario escribió un número erróneo/falso en el registro, nos perderíamos de poder conectarlo con sus futuros contactos vía ese número.

[5] Ver función `tag_weekday` en script_mensajes.py para detalles de la implementación.
[6] Ver función `tag_collections_cycle` en script_mensajes.py para detalles de la implementación.
[7] Ver función `assign_tenure` en script_mensajes.py para detalles de la implementación.
[8] Para este cálculo de mínima se toman como referencia 1 código provincial por provincia, aunque puede haber varios por cada provincia, y una longitud de números de celular de por lo menos 6 dígitos.

b) *Aun insertando el número correcto al momento del registro, el cliente podría contactarse desde otro número de teléfono*. Es cierto, en este caso parte del poder explicativo de los *features* se irá ya que se decidió dejar en blanco los *features* contextuales cuando no se pueda *matchear* el número de teléfono de la base *nx_clientes* con el número de teléfono de contacto. De esta forma, queda asegurado qué solo se *matchean* teléfonos iguales, y no se toma ningún supuesto para asumir conexiones entre ambas bases más allá del número telefónico.

c) *Otra persona podría contactarse desde el número de teléfono registrado, aun cuando no estén consultando por la cuenta registrada, sino por otra.* A modo de ejemplo, esto podría suceder si alguien utiliza el teléfono de un pariente como medio de contacto con atención al cliente, pero hace un reclamo por la propia cuenta, registrada con otro número telefónico. Esto podría suceder, sin embargo, no es esperable que contenga una gran representatividad, ya que, el acceso directo al canal de atención al cliente de WhatsApp se encuentra dentro de la aplicación, la cual contiene información sensible como ser movimientos de caja de ahorro, saldos, consumos con tarjeta de crédito. Es relativamente seguro tomar como supuesto que habrá un alto porcentaje de contactos que sean del propio dueño del celular y el propio cliente registrado en la aplicación.

Figura 9: detalle de reducción de información en el procesamiento de las bases



A continuación, se describen algunos de los pasos realizados en la implementación para cuantificar la perdida de información:

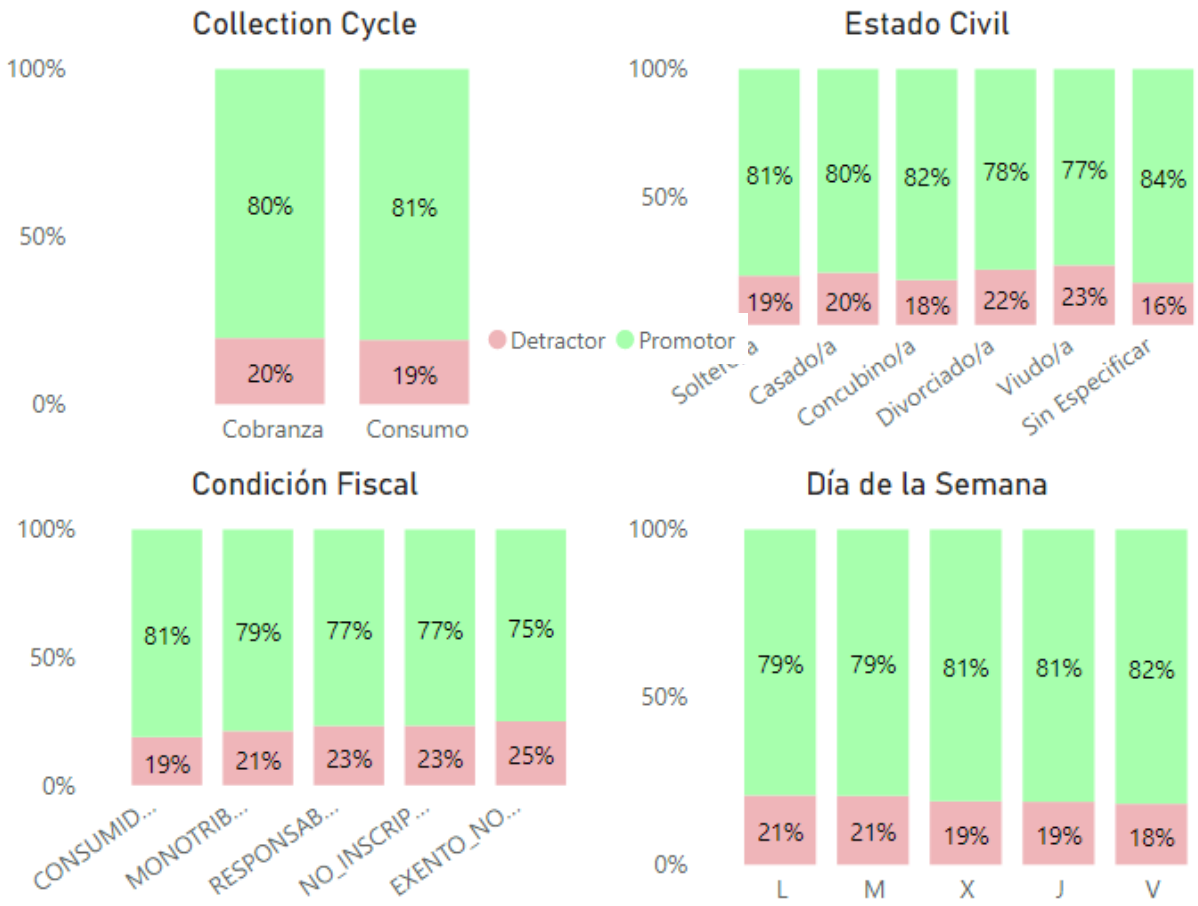
En el proceso se descrito se realizó la remoción de duplicados de número de teléfono tanto para la base de mensajes como de clientes, incluyendo la remoción de filas con dato vacío en *nx_clientes*. Esto implicó una reducción del 22,77% de la base de *nx_clientes.*

También, para evitar inconsistencias detectados en ambas bases respecto de diferencias en cómo se registra el código de área (por ejemplo, celulares de Buenos Aires registrados como 5491153882555 en una base y como 01153882555 en otra base), se normalizaron los teléfonos en ambas bases dejando únicamente los últimos 9 caracteres. Esto resultó en una reducción de sólo el 0,02% de la base de *nx_clientes* y una reducción del 0,00% de la base de *bm_mensajes*, mostrando la robustez de la normalización del número de teléfono a los últimos 9 caracteres.

Como resultado final, se termina con una base unificada de mensajes y datos contextuales que contiene 163.112 números de teléfono únicos, que se guardan en el *dataframe train_data_features*.

Ya con este set de *features* creados, se expande a continuación el análisis exploratorio para identificar relaciones entre estos *features* y la variable dependiente, con el fin de orientar sobre el poder predictivo que puedan tener.

Figura 10: distribución de promotores para los distintos valores de las variables Collection Cycle*, Estado Civil, Condición Fiscal y Día de la Semana*



En estos gráficos se observa la distribución del *mix* de promotores para la apertura de algunos de los *features* categóricos generados. Recordemos que, en el set de entrenamiento, la proporción de promotores es del 80%.

De un vistazo pareciera no haber grandes diferencias en el *mix* de promotores en ninguno de los cortes. En lo que son los *features* de *Collections Cycle* y Día de la semana, el rango entre el mínimo y el máximo es de 3 pp. En lo que son los *features* de Estado Civil y Condición Fiscal pareciera haber un poco más de rango entre el mínimo y el máximo (7 pp en Estado Civil y 6 pp en Condición Fiscal). Aun así, las categorías en las que se da la mayor amplitud ("Viudo/a" y "Sin especificar" para Estado Civil; "EXENTO_NO_GRAVADO" para Condición Fiscal) son categorías poco representativas, lo que reduce la confianza del rango entre el mínimo y el máximo, ya que representan el 1,0%, 0,4% y 0,1% de sus *features* respectivamente

Sección 7 – *Features* conversacionales

La generación de *features* conversacionales anexa a *train_data_features* una serie de *features* que están relacionados con cómo se dan las características de la conversación. En específico, lo que se busca en estos *features* es capturar características de la dinámica de la conversación que puedan contener algún indicio de la molestia (o satisfacción) señalizada por indicadores de tiempo de la conversación y de forma de conversar de ambas partes: agente y cliente. De esta forma se espera poder detectar características de tiempos y formas de hablar que se asocien a estados de satisfacción o insatisfacción de los clientes. Esto podría ilustrarse con dos ejemplos muy simples:

a) Se esperaría que esperas largas en la primera respuesta del agente desde que el cliente es derivado disminuyan el nivel de satisfacción con el agente que atiende, siendo entonces este tiempo de espera un potencial predictor de la nota final con la que se evalúa al agente.
b) Se esperaría que conversaciones con mayor cantidad de mensajes enviados por ambas partes puedan ser reflejo de conversaciones más complejas, y posiblemente más tediosas para el cliente, pudiendo ser el número de mensajes totales enviado en la conversación un potencial predictor de la nota final con la que se evalúa al agente.

A continuación, se describen todos los *features* conversacionales extraídos:

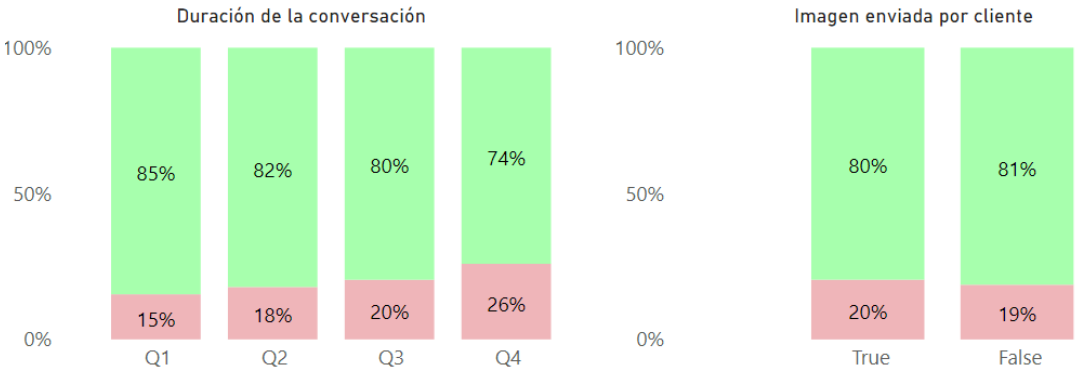
Tabla 5: descripción de los features *conversacionales*

Función que extrae el *feature*	**Descripción del *feature* extraído**
extract_conv_duration	Tiempo (en minutos) de una conversación
image_sent_tag	Indica si el cliente envió alguna imagen durante la conversación
wait_offer_count	Indica cuántas veces se le ofreció a un cliente continuar esperando[9]
msg_per_conv_agente	Indica la cantidad de mensajes enviados por el agente en una conversación
msg_per_conv_cliente	Indica la cantidad de mensajes enviados por el cliente en una conversación
msg_per_block_agente	Indica la cantidad de mensajes por bloque promedio enviados por el agente en una conversación
msg_per_block_cliente	Indica la cantidad de mensajes por bloque promedio enviados por el cliente en una conversación
msg_per_conv_total	Indica la cantidad de mensajes totales enviados por agente y cliente en una conversación
average_delay	Tiempo de la máxima espera (en minutos) y de la espera promedio (en minutos) entre todas las esperas por respuesta que tiene un cliente, medidas desde el momento que es derivado o envía el último mensaje de un bloque, hasta el tiempo del siguiente mensaje de un agente.

A continuación, se muestra una profundización en el análisis exploratorio de algunos de los *features* en relación a la variable dependiente, para ganar una orientación en el potencial poder explicativo de las mismas. Para algunas variables numéricas, se describan los valores promedio para cada cuartil de la variable, de forma tal de poder analizar el rango inter cuartil del *mix* de promotores.

[9] Cuando un cliente es derivado a agentes, pero se encuentran todos ocupados, queda en una cola de espera hasta ser asignado. Mientras el cliente se encuentra en la cola de espera y no haya sido atendido por un agente, se le pregunta con cierta frecuencia si quiere continuar esperando, a lo cual puede responder que sí o que no. Lo que hace la función *wait_offer_count* es contabilizar la cantidad de veces que se le ofrece continuar esperando a un cliente.

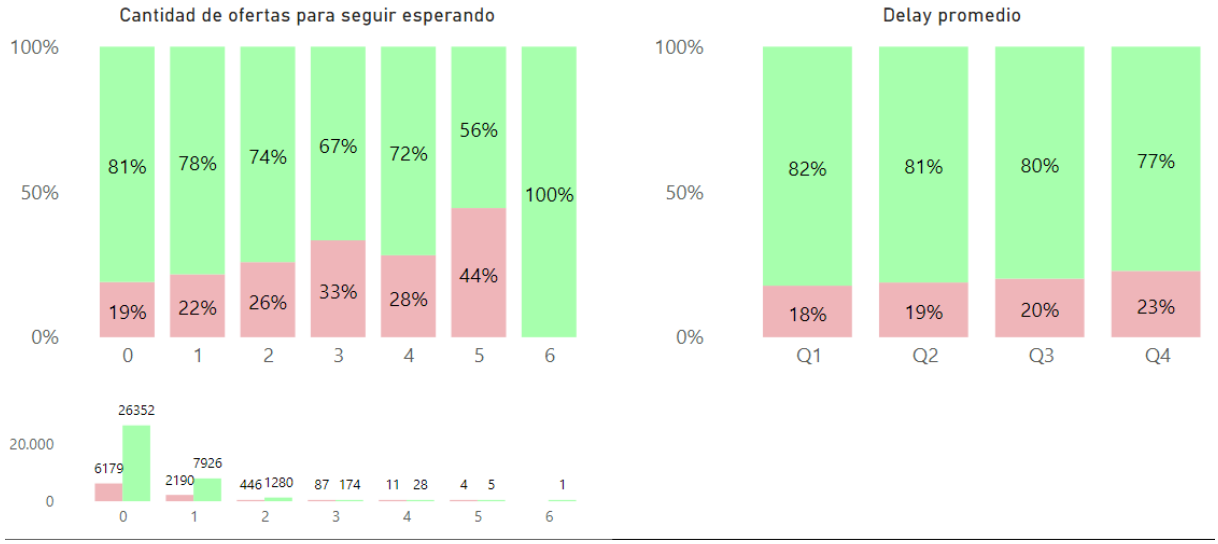
Figura 11: distribución de promotores para los distintos cuartiles de las variables Duración de la Conversación e Imagen Enviada por Cliente



En lo que respecta a la Duración de la Conversación, se observa un rango inter cuartil de 11pp, lo que podría orientarnos a que tenga cierto poder explicativo, mientras que, en la variable de Imagen enviada por cliente, este se reduce a 1pp, lo que nos sugiere que no es una variable con alto poder predictivo.

A continuación, se muestra el *mix* de promotores para dos variables más, y debajo se muestra su representatividad en los datos de entrenamiento:

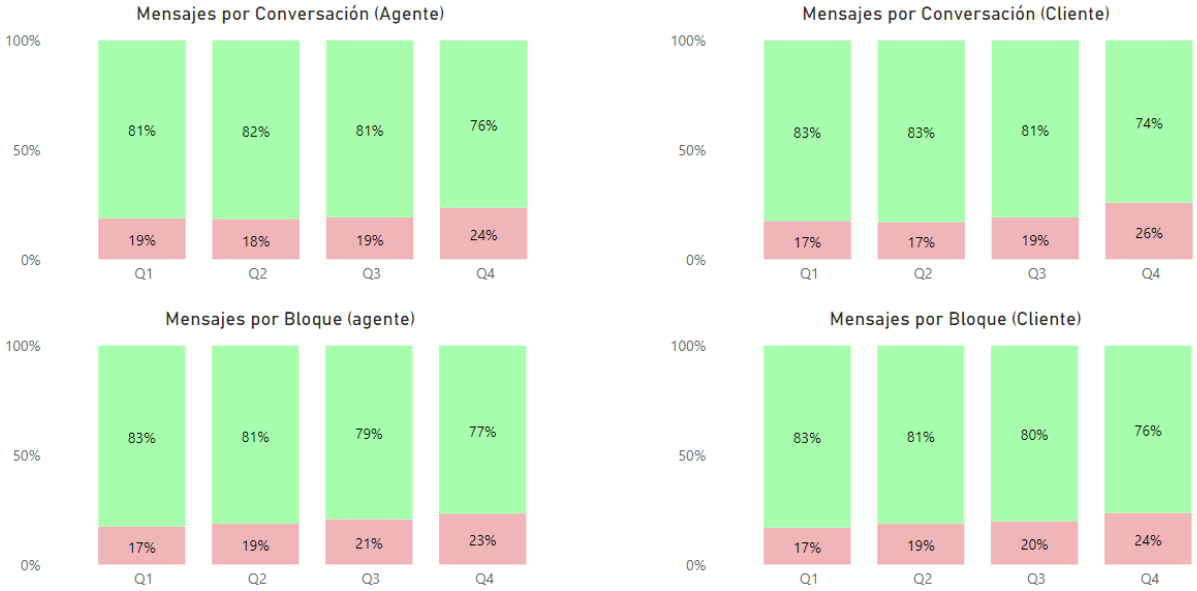
Figura 12: distribución de promotores para los distintos valores y cuartiles de las variables Cantidad de Ofertas para Seguir Esperando y Delay *Promedio*



Pareciera haber una tendencia más clara en la variable *wait_offer*. Sin embargo, cuando vemos la representatividad de las conversaciones con valores mayores o iguales a 2, vemos que cae significativamente, representando en su conjunto sólo el 4,5%. Por lo que el rango inter cuartil más representativo es entre los valores 0 y 1, que cae a 3pp.

Continuando con el análisis, ahora de las métricas de cantidad de mensajes por bloque/conversación para agente y cliente:

Figura 13: distribución de promotores para los distintos cuartiles de las variables Mensajes por Conversación (agente), Mensajes por Conversación (Cliente), Mensajes por Bloque (agente) y Mensajes por Bloque (cliente)



En este caso observamos que los rangos inter cuartiles máximos para las variables de Mensajes por Conversación y Mensajes por Bloque se dan en el cliente, teniendo un rango inter cuartil de 9pp y 7pp respectivamente[10].

A continuación, un resumen de los rangos máximos por categoría y rangos inter cuartiles del *mix* de promotores para cada *feature*, incluyendo tanto los *features* conversacionales como los contextuales, considerando las categorías con representatividad de al menos 5% para calcular los rangos:

[10] En los casos de distribuciones en los que no se detalla la cantidad absoluta en cada categoría/cuartil, es con el fin de no recargar la visualización, ya que estos *features* tienen representatividad alta en todos los cuartiles/categorías expuestos.

Tabla 6: rango inter cuartil o rango entre mínimo y máximo de mix *de promotores para los distintos* features *contextuales y conversacionales*

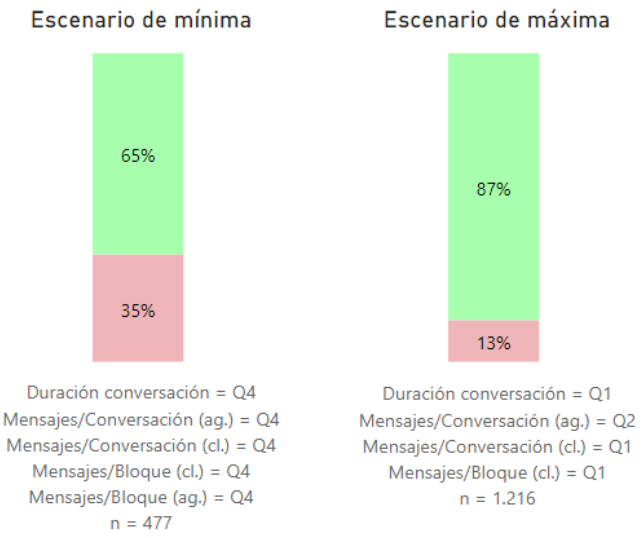
Tipo	*Feature*	Mínimo / Q1	Máximo / Q4	Rango
Contextual	*Collection Cycle*	80% (Cobranza)	81% (Consumo)	**1pp**
Contextual	Estado Civil	78% (Divorciado)	82% (Concubino)	**4pp**
Contextual	Condición Fiscal	79% (Monotributista)	81% (Consumidor Final)	**2pp**
Contextual	Día de la semana	79% (Lunes)	82% (Viernes)	**3pp**
Conversacional	Duración conversación	74% (Q4)	85% (Q1)	**11pp**
Conversacional	Imagen enviada	80% (True)	81% (False)	**1pp**
Conversacional	N° de ofertas para seguir esperando	78% (1 oferta)	81% (0 ofertas)	**1pp**
Conversacional	*Delay* promedio respuesta	77% (Q4)	82% (Q1)	**5pp**
Conversacional	Mensajes/Conversación (agente)	76% (Q4)	82% (Q2)	**6pp**
Conversacional	Mensajes/Conversación (cliente)	74% (Q4)	83% (Q1)	**9pp**
Conversacional	Mensajes/Bloque(agente)	77% (Q4)	83% (Q1)	**6pp**
Conversacional	Mensajes/Bloque (cliente)	76% (Q4)	83% (Q1)	**7pp**

Algunas conclusiones que se desprenden de estos datos, que luego serán o no validadas por el modelo predictivo:

- Las variables conversacionales parecieran contener *features* con mayor poder explicativo que las contextuales, lo que sugiere una importancia mayor del momento de la conversación comparada a cómo llega el cliente a la conversación.
- Dentro de las variables conversacionales, la forma de hablar del cliente pareciera tener mayor relevancia que la forma de hablar del agente, esto pensando únicamente en el mayor rango que presentan los mensajes/conversación y mensajes/bloque para el cliente vs. el agente.
- Pareciera que hay un efecto en la misma dirección sobre la probabilidad de tener un promotor en la "intensidad" del agente y del cliente, entendiendo la intensidad como la cantidad de mensajes por bloque y por conversación: cuando esta intensidad es muy alta por parte del cliente, se reduce el *mix* de promotores de la muestra. Para el caso de los agentes, pareciera suceder algo similar, pero hay algún indicio de un *sweet spot* de esta "intensidad", evidenciado en que el máximo de promotores de mensajes/conversación para los agentes no se da en el Q1 sino en el Q2, sugiriendo que mensajes muy escuetos de los agentes, tampoco son el óptimo.

Resulta tentador hacerse la pregunta de qué sucede si combinamos todos estos efectos a la vez. Lamentablemente la combinación de todos los efectos, al ser la intersección de muchas variables, deja un número muy chico de observaciones. Pero a continuación se muestra el análisis considerando filtrar los datos con las combinaciones para los *features* que presentan el mayor rango manteniendo un n > 400:

Figura 14: mix *de promotores para la muestra seleccionando los valores de* features *de mínima y de máxima*



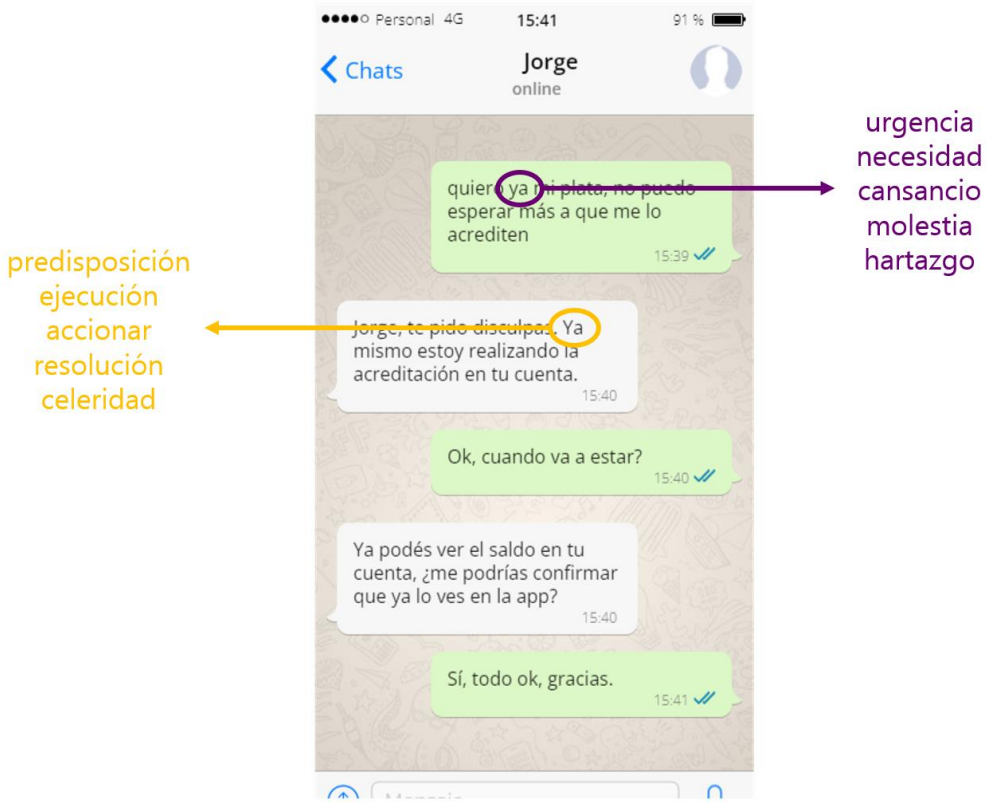
Puede observarse que al combinar el efecto de las variables descritas (todas provenientes de *featues* conversacionales) se logra ampliar la diferencia en el *mix* de promotores, llegando a un rango entre ambos escenarios de 22pp, que resulta el doble del rango máximo que una variable sola podía alcanzar (Duración conversación = 11pp). Pareciera a priori haber algo de poder explicativo aquí.

Sección 8 – *Features* semánticos

Para la generación de los *features* de carácter semántico se decidió incorporar *features* que puedan señalar y diferenciar la frecuencia con el que se encuentran presentes algunas palabras dentro de las conversaciones, entendiendo que la connotación o la semántica de esas palabras, en el contexto de un contacto con un servicio de atención al cliente, pueden ser indicativas de la emocionalidad de la conversación e incluso del nivel de percepción del cliente respecto del agente. Por este motivo, se espera que puedan brindar poder explicativo respecto a la evaluación que hará el cliente al responder la encuesta de satisfacción.

Cabe destacar, que la connotación de las palabras utilizadas puede cambiar enormemente dependiendo de quién sea el emisor de estas palabras. Asimismo, la dirección en la cual esa palabra "empuje" la probabilidad de que la conversación resulte en un puntaje promotor, también puede variar diametralmente dependiendo del emisor. Veamos la siguiente conversación para ejemplificar estos conceptos:

Figura 15: chat de referencia evidenciando el uso de una misma palabra con distinta connotación según el emisor



En esta conversación se observa el uso de la palabra "ya". En esta conversación busca ilustrarse como la misma palabra puede transmitir emociones distintas. Uno de los factores que podría distinguir estas distintas connotaciones de la palabra es el emisor de la misma. En el ejemplo, el cliente utiliza la palabra ya pare mostrar su necesidad inmediata y urgente de su dinero. Podría estar reflejando también hartazgo, molestia o cansancio, si es que indica que no quiere perder más tiempo del ya perdido en contactos anteriores.

Por el contrario, la utilización de la palabra ya por parte del agente muestra un significado y una emocionalidad distintos. En este caso, el agente está intentando transmitir su compromiso con la urgencia del cliente, y su predisposición a ejecutar la acción de forma rápida, mostrando que va a brindar una resolución al problema.

En la implementación realizada para capturar los *features* semánticos, se tiene en cuenta esta diferencia de los emisores del mensaje para poder capturar como "palabras distintas" aquellas que fueron dichas por agentes y aquellas que fueron dichas por clientes.

Entonces, para poder capturar la frecuencia de utilización de las palabras por parte de agentes y clientes, se implementó la sección 8: *Features* semánticos, en *script_mensajes.py*, a partir de 4 funciones.

➔ **Definir *stopwords***

En la función `define_stopwords` se definen las *stopwords* a utilizar, que serán quitadas por tener un bajo poder predictor de la nota final que pondrá el cliente. Se utilizan las *stopwords* en español de Natural Language Toolkit (nltk.org) del idioma español. Para hacer un refinamiento de estas *stopwords*, se hace una interpretación una por una de todo el listado, y se terminan eliminando de las *stopwords* algunas palabras que podrían tener algún poder explicativo en el contexto de una conversación de atención al

cliente, por ejemplo, las palabras “pero”, “muchos”, “mío”[11]. Asimismo, se agregan a las *stopwords* algunas palabras que no aportan ningún valor como algunas preposiciones y algunos *tags* que se agregan de forma automática a la columna *mensaje* de la base datos *bm_mensajes* (para mayor detalle, ver Apéndice A: Descripción de los datos de origen

).

En el caso de existir alguna palabra que faltó incluir en los *stopwords*, en la sección 9 del código se realizará una normalización tf-idf que disminuirá la importancia relativa de esos *stopwords*.

➔ Concatenar mensajes

La implementación de las funciones `chain_msgs_client` & `chain_msgs_agent` se realiza con el objetivo de poder generar dos objetos del tipo diccionario (uno para clientes y uno para agentes) que contenga como clave el identificador único de la conversación (*conv_id*) y como valor un *string* con todas las palabras enviadas por los emisores (cliente y agente) en esa conversación. De esta forma, se extrae en un formato más amigable la información cruda de las palabras que dijeron ambos emisores, identificadas con un identificador de conversación. A continuación, un ejemplo del formato obtenido:

Figura 16: líneas de ejemplo del dataframe *obtenido tras la aplicación de la función* `chain_msgs_client`

Index	conv_id	messages
3182	3183	Hola buenas tardes quiero saber cuánto se …
3183	3184	Por qué no puedo abrir la app {"button":"N…
3184	3185	{"button":"Agente","entities":"{\"entity\"…

Como puede observarse, aparecen algunos de los *tags* mencionados anteriormente, que serán corregidos mediante la eliminación si están incluidos en las *stopwords*, o su efecto será minimizado mediante la normalización tf-idf debido a su alta ocurrencia en todas las conversaciones.

➔ Tokenización

La función `tokenize` toma como input el diccionario devuelto en las funciones recién descriptas y las *stopwords* definidas anteriormente. A partir de estos dos inputs genera una serie de acciones sobre el diccionario:

i. Reemplaza las vocales con tildes por vocales sin tildes.
ii. Separa los *strings* de texto para cada para *conv_id-emisor* en tokens únicos. Por ejemplo, para $(conv\_id = i,\ emisor = j),\ j \in (cliente, agente)$ se genera una lista cuyos elementos son cada palabra enviada en todos los mensajes de *j* en la conversación *i*.
iii. Se normalizan todas las letras a minúsculas.
iv. Se remueven las *stopwords*.

[11] En el contexto de atención al cliente de una fintech no es inusual encontrar adjetivos cuantificadores indefinidos como mucho, poco, demasiado, ya que suele hablarse de cantidades de dinero y puede expresar una noción del grado de importancia que se la da al monto de dinero del cual se habla.

De esta forma se obtiene, tanto para agentes como clientes, sendos *dataframes* que contienen en la primera columna un *conv_id* y en la segunda una lista de palabras:

Figura 17: líneas de ejemplo del dataframe *obtenido tras la aplicación de la función* `tokenize`

Index	conv_id	messages
3182	3183	['hola', 'buenas', 'tardes', 'quiero', 'sab…
3183	3184	['no', 'puedo', 'abrir', 'app', 'naranja', …
3184	3185	['agente', 'entity\\', 'class\\', 'stringen…

➔ **Rectangling**

Se implementan las funciones `data_rectangling_agentes` & `data_rectangling_clients` para remanejar los *dataframes* generados a partir de la función `tokenize`. El objetivo de esta función es convertir los *dataframes* al formato *document-term matrix*, en el que los "*documents*" se corresponde al id de conversación y los *terms* se corresponden a las palabras dichas por agentes y clientes.

A continuación, se describen los detalles de la implementación de las funciones de `data_rectangling`.

- Se determina un *threshold* de 100, que indica la frecuencia mínima con la que debe aparecer una palabra en **todo** el conjunto de entrenamiento (~10 millones de mensajes) para ser incluida como *feature* o columna en la *document-term matrix*. De esta forma se intenta reducir el *overfitting* y el aprendizaje sobre palabras que no aparecen con mucha frecuencia, colaborando también con el consumo de memoria del equipo en el que se implementa esta función.
- Haciendo uso del método `Counter`, de la librería `collections`, se genera un diccionario con el conteo de cada palabra
- Este diccionario es transformado a un *dataframe* con el método `from_dict` y se asigna en cada posición de la matriz el conteo del par *conv_id-palabra* correspondiente. Al nombre de la columna se le antepone el prefijo "cl_" o "ag_" dependiendo si la palabra fue dicha por un cliente o un agente.

Como resultado, estas funciones devuelven, tanto para los clientes como para los agentes, sendos *dataframes* de esta forma:

Figura 18: líneas de ejemplo del dataframe *obtenido tras la aplicación de la función* `data_rectangling_clients`*, mostrando el conteo de palabras por cada conversación*

Index	cl_acredito	cl_agente	cl_ah	cl_ahi	cl_ahora	cl_aparece	cl_aplicacion	cl_conv_id
0	nan	1	nan	nan	nan	nan	nan	1
1	nan	nan	nan	nan	nan	nan	nan	2
2	nan	nan	nan	nan	nan	nan	1	3
3	nan	nan	nan	nan	nan	nan	nan	4
4	nan	nan	nan	nan	2	nan	nan	5
5	nan	2	nan	nan	2	nan	nan	6
6	nan	nan	nan	1	1	nan	nan	7

Lo que se observa es una matriz que tiene una columna para cada palabra y una fila para cada conversación, identificada en la columna final por el id de la conversación. Lo que se observa en las celdas es el conteo de las palabras en la conversación. En este caso, estamos observando la *document-term matrix* para los clientes. Esta matriz nos refleja, por ejemplo, que en la conversación con *conv_id* = 6, el cliente dijo 2 veces la palabra "agente" y 2 veces la palabra ahora. Asimismo, se observa que la palabra "ahora" se nombró también en la conversación 5 (2 veces) y en la conversación 7 (1 vez).

Sección 9 – *Dataframe* final

En esta sección se busca hacer los arreglos finales a los *dataframes* generados y además se busca unirlos todos en un único *dataframe* que contenga a todos los *features* generados. Asimismo, en esta sección se realiza la normalización tf-idf de los *features* semánticos. A continuación, la descripción de los pasos realizados en esta sección.

- **Remoción de filas redundantes**. En el *dataframe* que contiene toda la información de los mensajes y los *features* conversacionales y contextuales, hay muchas filas para cada conversación (recordar que hay una fila por mensaje enviado), pero sólo se necesita 1 fila por conversación, que tenga las columnas correspondientes a los *features* de ésta. Para ellos se utiliza el método *drop*, para quitar todas las filas que no tienen el campo *score_agent_conv_id*, que, además de contener la información de a qué conversación corresponden todos los *features* de la fila, indica la fila en la que se registran todos los *features* de cada conversación.
- **Creación de matriz semántica**. Se realiza el *join* de las *document-term matrix* de clientes y agentes, usando como clave primaria el id de conversación (aparece como *cl_conv_id* en la matriz de clientes y como *ag_conv_id* en la matriz de agentes). Para simplificar, se unifican ambas columnas en una llamada *conv_id*. Los conteos nulos, se reemplazan por el número 0 para preparar la matriz para la normalización tf-idf.
- **Normalización tf-idf**. Mediante la función *TfidfTransformer* de la librería de *feature extraction* de *Sci-Kit Learn* (scikit-learn.org) se realiza la normalización de la matriz semántica con la normalización tf-idf. Lo que se busca en esta implementación es transformar el conteo de palabras de la matriz semántica en un número que sea una normalización de esa frecuencia. Lo que se busca es reflejar el producto entre dos factores:
    a) la frecuencia de cada palabra es normalizada (*tf*) para cada conversación (c) y para cada palabra (p) con la siguiente función:

$$tf_{(c,p)} = \begin{cases} 1 + \log_{10} count(p,c) & si \quad count(p,c) > 0 \\ 0 & de\ otra\ forma \end{cases}$$

b) la inversa de la frecuencia de las palabras en los documentos (*idf*) para la palabra p, se calcula según la siguiente fórmula:

$$idf_p = \log\left(\frac{N}{cf_p}\right)$$

Donde:

N: $cantidad\ total\ de\ conversaciones$
cf_p: $frecuencia\ de\ aparición\ en\ documentos\ de\ la\ palabra$ p

A partir de estos números se obtiene la frecuencia de cada palabra, ya normalizada por la normalización tf-idf, obteniendo una matriz de este estilo:

Figura 19: líneas de ejemplo del dataframe *obtenido tras la aplicación de la función* `tf_idf`

Index	ag_ayudar	ag_ayudarte	ag_ayudo	ag_baja
121		0.147266	0	0
122		0.0643627	0	0
123		0.10465	0	0
124	0892156	0	0	0
125		0.0626035	0	0.105518
126		0.117702	0	0

➔ **Binarización variable dependiente**. En el último paso antes de anexar la matriz semántica a la matriz de mensajes ya reducida a 1 línea por conversación, se ejecuta la binarización de las notas de la encuesta respondida por los usuarios, es decir, la variable dependiente. El criterio que se utilizó fue el siguiente:
- Notas del 1 al 4: se asigna un 0 (detractor)
- Nota 5: se asigna un 1 (promotor)

En los modelos predictivos que a continuación se explican, se aprenderá a predecir promotores (1's).

¿Por qué se elige este umbral? En este caso tiene que ver con utilizar un criterio que sea al mismo tiempo exigente y comparable con los criterios de la industria de atención al cliente en general.

Respecto a la exigencia: este umbral es el más exigente ya que clasifica como promotores únicamente a aquellos que hayan dado la máxima nota. El resto, ya sea que hayan calificado un 1, 2, 3 o 4, serán considerados todos detractores. Esto permite, a la hora de comunicar los resultados y posibles aplicaciones de este trabajo, asegurar que los criterios tomados tienen un estándar de alta exigencia.

Respecto a los criterios de la industria: la elección de la nota 5 no es casualidad, así como tampoco lo es la elección de las palabras "promotor" y "detractor". En la industria una de las más (sino la más) conocida métrica para medición de experiencia es el NPS (*Net Promoter Score*). Como se explica en el libro de quien creó esta métrica[12], en una escala del 1 al 10, se considera promotores a quienes puntúen 9 o 10. Como las notas 9 y 10 representan un corte que deja al 20% de las notas más altas como

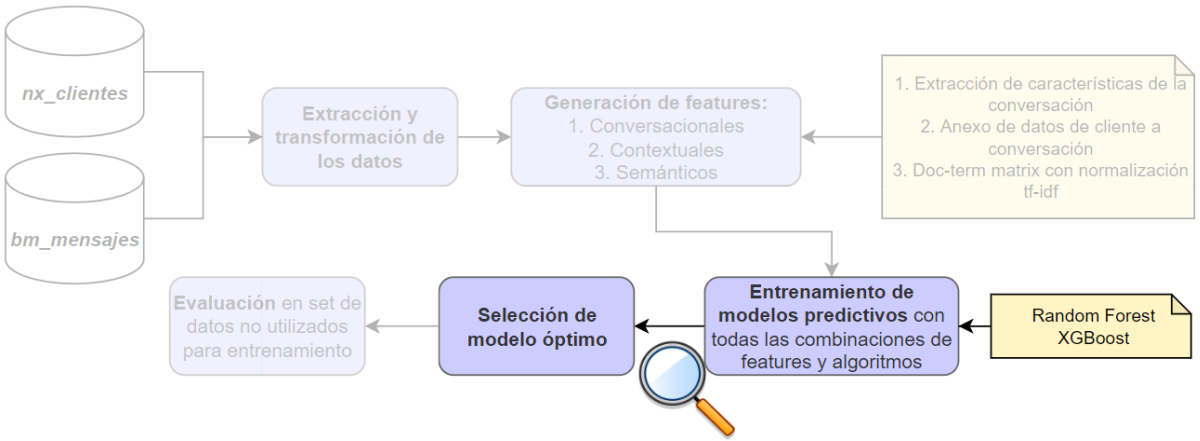
[12] Fred Reichheld, La Pregunta Decisiva 2.0, LID Editorial

promotores, se utiliza ese mismo criterio adaptado a la escala del 1 al 5 con la cual se puntúa en la empresa estudiada.

- ➔ **Unión de *datasets***. Como paso final, se hace la unión de los *datasets*, anexando la matriz semántica normalizada y obteniendo un *dataframe* final que se utilizará para el entrenamiento de los modelos predictivos de 45.003 x 8.221 (conversaciones x *features*).

3.4 Entrenamiento de modelos predictivos

En esta sección se detallará el proceso de entrenamiento de los modelos para la predicción de la variable dependiente:



En esta etapa, se busca hacer un entrenamiento sobre distintos algoritmos para poder predecir, a partir del *dataframe* generado la variable dependiente que toma valor 1 si la conversación resulta en un puntaje de 5 (promotor) y 0 si la conversación termina en un puntaje menor a 5 (detractor). Para ello, se trabajará sobre distintos *subsets* de los datos, cada uno con distintos conjuntos de *features* (conversacionales, contextuales y semánticos) "encendidos". De esta forma se evaluarán por validación cruzada en datos de entrenamiento todas las combinaciones de *features* con los 2 algoritmos seleccionados: *Random Forest* y *XGBoost*[13].

A continuación, se describen las secciones detalladas en el script *modelos.py*, que reflejan los pasos seguidos en esta etapa de entrenamiento de modelos predictivos.

<u>Sección 10 – Preparación de *datasets* para algoritmos</u>

En esta sección se genera se generan todos los *subsets* de datos que se utilizarán como datos de entrenamiento en los 2 algoritmos nombrados. De esta forma, lo que se busca es obtener la siguiente cantidad de combinaciones de modelos:

$$Cantidad\ modelos = (2^f - 1) \times a$$

Donde:

$f: cantidad\ de\ categorías\ de\ features$
$a: cantidad\ de\ algoritmos\ a\ utilizar$

En este caso $f = 3$ ya que contamos con 3 categorías de *features*: conversacionales, contextuales y semánticos. Mientras que $a = 2$, ya que se utilizarán 2 algoritmos: *Random Forest* y *XGBoost*.

Con este fin se implementa la función `gen_dataset_combination` en el script *modelos.py*. Para facilitar la implementación de esta función, se crean dos listas, *col_context* y *col_conversational*, que contienen los nombres de las columnas del *dataframe train*, que se corresponden con los nombres de

[13] Este tipo de algoritmos de clasificación es parte de los algoritmos sugeridos en Mohammad, K. S., & Hemmatian, F. (2019). *An efficient preprocessing method for supervised sentiment analysis by converting sentences to numerical vectors: A twitter case study. Multimedia Tools and Applications*, en este caso para la clasificación de sentimientos de comentarios en Twitter.

los *features* del *dataframe train* correspondientes a los conjuntos de *features* contextuales y conversacionales respectivamente. Por complementariedad, los restantes *features* pertenecerán al conjunto de *features* semánticos.

En la función `gen_dataset_combination`, se realiza el siguiente proceso:

- ➔ Las variables categóricas se convierten al tipo *category*.
- ➔ Para cada combinación de *features* se "recorta" el *dataframe* original para quitar las columnas que no sean necesarias.
- ➔ Se hace un *split* en los *subsets*, para tener por un lado las columnas de los *features* (nomenclatura: *nombre_subset_x*) y por otro lado la columna de la variable dependiente (nomenclatura: *nombre_subset_y*)

Como resultado, la función devuelve el *dataframe* original con las variables categóricas transformadas al tipo *category* y los 14 *dataframes* correspondientes a las distintas combinaciones de *features*, con sus correspondientes *dataframes y*, con el resultado de la variable dependiente, como puede observarse en la siguiente tabla:

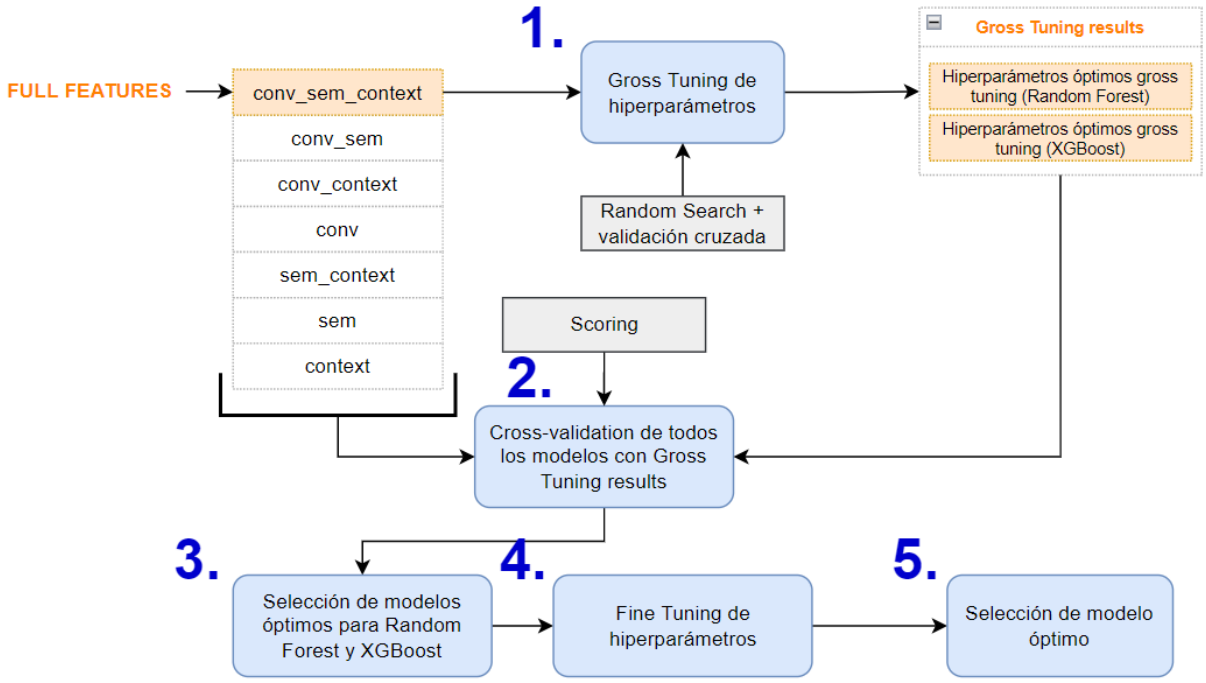
Tabla 7: cantidad de features *y nomenclatura por cada* subset

***Features* presentes**	**Nomenclatura**	**Cantidad de *features***
Conversacionales + semánticos + contextuales	*conv_sem_context_x* *conv_sem_context_y*	8123
Conversacionales + semánticos	*conv_sem_x* *conv_sem_y*	8112
Conversacionales + contextuales	*conv_context_x* *conv_context_y*	21
Conversacionales	*conv_x* *conv_y*	10
Semánticos + contextuales	*sem_context_x* *sem_context_y*	8113
Semánticos	*sem_x* *sem_y*	8102
Contextuales	*context_x* *context_y*	11

Antes de avanzar a la siguiente sección, vale la pena aclarar cuál será la estrategia a seguir en lo referido al entrenamiento y refinamiento de hiperparámetros de los modelos.

La estrategia elegida se resume en el siguiente esquema:

Figura 20: esquema con la estrategia para el refinamiento de hiperparámetros y selección de modelo óptimo



1. Sobre el modelo que contiene los 3 grupos de *features* (de ahora en más, modelo *full features*), se realiza un *gross tuning* de los hiperparámetros para los algoritmos de *XGBoost* y *Random Forest*. Se utiliza un *random search* con validación cruzada para la selección de los mejores hiperparámetros.
2. Con los resultados de 1., se evalúan en los datos de *train* los restantes 6 modelos, en ambos algoritmos, y se obtiene el score para cada uno de estos modelos.
3. En bases al criterio de score seleccionado, se elige 1 modelo óptimo para *Random Forest* y 1 modelo óptimo para *XGBoost*
4. Sobre los 2 modelos óptimos elegidos en 3., se realiza un *fine tuning* de los hiperparámetros
5. Se evalúa el score de los modelos y se elige el modelo óptimo.

Sección 11 – *Gross tuning* de hiperparámetros

Siguiendo con la estrategia descripta anteriormente, en el código se implementa la función `x_normalization`. Esta función toma como input cualquiera de los *subset* con el sufijo "_x" de la Tabla 7.

Lo que hace esta función es utilizar las funciones *OrdinalEncoder* y *StandardScaler* de la librería *preprocessing de Sci-Kit Learn* (scikit-learn.org) para:

a) codificar las variables categóricas de forma tal que puedan ser tomadas como input para los algoritmos de aprendizaje,
b) normalizar las variables por su media y desvío estándar de forma tal que no existen un desbalance en el valor predictivo de las variables cuando los algoritmos aprendan, debido a sus órdenes de magnitud.:

➔ *fit* (*StandardScalerobject*): indica si la codificación y normalización deben realizarse aprendiendo los parámetros del *dataset* provisto o no

- ➔ *scaler* (bool): permite agregar parámetros externos para codificar y escalar al *dataset* provisto (se utilizará para los datos de *test*)
- ➔ *return_norm_sc* (bool): indica si la función debe retornar o no un *StandardScalerobject* con los parámetros aprendidos al hacer *fit* del *dataset* provisto

De esta forma, en este paso se realiza la normalización con los datos de entrenamiento (*fit* = True) y se guardan los parámetros aprendidos para su posterior uso (*return_norm_sc* = True).

A continuación, se realiza la búsqueda aleatoria de hiperparámetros para ambos algoritmos y para la versión *full features* en las funciones implementadas llamadas `random_grid_rf` y `random_grid_xgb`.

Para ello, se realizan 20 iteraciones y se utiliza *cross-validation* en 5 *folds* para testear los distintos hiperparámetros. La matriz de hiperparámetros sobre la cual se realiza la búsqueda y los resultados óptimos (*gross tuning results*) se muestran a continuación:

Tabla 8: resultados del proceso de gross tuning

Algoritmo	Hiperparámetro	Rango de búsqueda	*Gross tuning results*
Random Forest	*n_estimators*	15 valores entre 25 y 1.000	**651**
	max_features	['sqrt', 'auto']	**sqrt**
	max_depth	[5, 40, 75, 110, 155, 190]	**155**
	min_samples_split	[4, 8, 12, 16]	**12**
	min_samples_leaf	[2, 4, 6]	**2**
	bootstrap	[True, False]	**False**
	criterion	['gini', 'entropy']	**Gini**
XGBoost	*learning_rate*	[0.01, 0.05, 0.01, 0.2, 0.3, 0.5, 0.7]	**0.05**
	max_depth	[5, 40, 75, 110, 155, 190]	**5**
	gamma	[i/10.0 for i in range(0,5)]	**0.4**
	colsample_bytree	[0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	**0.3**
	reg_alpha	[1e-2, 0.1, 1, 10, 100]	**1**
	reg_lambda	[1e-2, 0.1, 1, 10, 100]	**100**
	subsample	[0.2, 0.4, 0.6, 0.8]	**0.6**
	n_estimators	[50, 100, 150, 250, 400, 600, 1000]	**100**

Sección 12 – *Cross-validation* en todos los modelos

En esta sección del código, se realiza la evaluación por *cross-validation* de los 7 modelos propuestos para ambos algoritmos, para un total de 14 combinaciones. Para ello, en esta sección se utilizan los parámetros obtenidos en la sección anterior (*gross tuning results*).

Se define un diccionario con el *scoring* a recolectar en la validación cruzada. A través de la función *make_scorer* de la librería de *metrics* de *Sci-Kit Learn* (scikit-learn.org) se crea el diccionario con las siguientes métricas:

- ➔ *Accuracy*
- ➔ *Precision*
- ➔ *Recall*
- ➔ *F1 score*

Para cada uno de los 14 modelos se realiza la normalización de los datos descripta (recordar que cada uno de los 7 *subsets* tiene distintas columnas) y se guardan los parámetros de la normalización para su posterior uso con los datos de *test*.

Se define para el *split* de los *folds* de *cross-validation* un *Stratified K-fold* de 5 splits y se utiliza la función *cross_validate* de la librería de *model_selection* de *Sci-Kit Learn* (scikit-learn.org) para la validación cruzada, asignando las métricas generadas anteriormente como parámetro de *scoring*.

De esta forma, se obtienen 4 scores para cada uno de los 14 modelos, cuyos resultados pueden observarse en la sección Resultados de la página 46.

Sección 13 – *Fine tuning* de mejores modelos

Luego de la sección anterior, se obtienen como resultados dos mejores modelos: uno para el algoritmo *Random Forest* y otro para el algoritmo *XGBoost*. Siguiendo con el paso 4. del esquema propuesto anteriormente, se procede al refinamiento de los hiperparámetros de estos dos modelos.

Para el refinamiento de los hiperparámetros, se crea una grilla de hiperparámetros para *Random Forest* y para *XGBoost*, con un rango más acotado (±10%) alrededor de los parámetros óptimos del *gross tuning*. Para la elección de los hiperparámetros óptimos se utiliza validación cruzada de 5 *splits*, determinada por el parámetro *cv* de la función *GridSearchCV* de *Sci-Kit Learn* (scikit-learn.org). La elección del algoritmo óptimo se realiza al *F1-score* óptimo de esta validación cruzada. Algunos de los factores, se dejan fijos según los resultados del *gross tuning* dado el crecimiento exponencial de combinaciones que implica y la demora excesiva según el equipo usado para este trabajo. De esta forma, se obtienen los siguientes resultados:

Tabla 9: resultados del proceso de fine tuning

Algoritmo	Hiperparámetro	Rango de búsqueda	*Fine tuning Results*
Random Forest	*n_estimators*	651	**651**
	max_features	['sqrt']	**sqrt**
	max_depth	[141, 150, 159, 168]	**159**
	min_samples_split	[10, 11, 12, 13]	**10**
	min_samples_leaf	[1, 2]	**2**
	bootstrap	[False]	**False**
	criterion	['gini']	**Gini**
XGBoost	*learning_rate*	[0.03, 0.05, 0.07]	**0.07**
	max_depth	[3, 4, 5, 6]	**6**
	gamma	[0.4]	**0.4**
	colsample_bytree	[0.3]	**0.3**
	reg_alpha	[10e-1.1, 10e-1, 10e-0.9]	**10e-1.1**
	reg_lambda	[10e1.9, 10e2, 10e2.1]	**10e1.9**
	subsample	[0.6]	**0.6**
	n_estimators	[100]	**100**

Luego de revisar los resultados por validación cruzada (ver sección Resultados en página 46), se selecciona finalmente el algoritmo óptimo a utilizar con sus hiperparámetros ya optimizados. El criterio de evaluación para seleccionar el modelo y los resultados, se detallarán en la siguiente sección.

4. Resultados

En esta sección se muestran los resultados de los procesos realizados para la optimización del modelo predictivo y, finalmente, los resultados finales de *performance* en los datos de *test*.

4.1 Criterio de evaluación

Para decidir el criterio de evaluación, es necesario tener en cuenta el contexto real del problema a resolver. El problema está relacionado con la predicción de promotores en las interacciones con atención al cliente. Para eso, se presenta a continuación una matriz de confusión teórica para entender cada casuística de la matriz[14]:

Tabla 10: matriz de confusión teórica

		Valor real	
		1 (Promotor)	0 (Detractor)
Predicción	1 (Promotor)	Clientes promotores predichos como promotores (TP)	Clientes detractores predichos como promotores (FP)
	0 (Detractor)	Clientes promotores predichos como detractores (FN)	Clientes detractores predichos como detractores (TN)

A los fines de poder decidir una métrica que capture de la mejor forma los objetivos relevantes de este modelo predictivo, se plantean dos casos de uso relevantes para la empresa en términos de la aplicación de este modelo:

i. Poder aprender cuáles son las buenas prácticas conversacionales de forma correcta, para replicarlas
ii. Poder capturar todas las buenas prácticas conversacionales para no perderse de ninguna

Este es un enfoque basado en potenciar los promotores.

Para ellos se decide utilizar la métrica del *F1 score*, que captura ambos objetivos, definida como:

$$F1\ score = \frac{2\ \times precision\ \times recall}{precision + recall}$$

Donde:

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

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

Maximizando el *recall*, entonces, logra perseguirse los puntos i. y ii. al considerar tanto precisión como *recall*.

Es verdad que podrían existir otros objetivos para la empresa, como por ejemplo la detección de detractores y buscar minimizar la cantidad de detractores que se "escapan". Ese sería otro enfoque de maximizar la detección de detractores; pero se considera que, en parte, la métrica propuesta para

[14] En la matriz se hace referencia a los conceptos Verdaderos Positivos, Verdaderos Negativos, Falsos Positivos y Falsos Negativos como TP, TN, FP, y FN respectivamente.

maximizar captura en parte esto, al maximizar el TPR, que minimiza la cantidad de falsos positivos en relación al total de positivos predichos.

4.2 Resultados de *gross tuning*

A continuación, se muestran los resultados en la validación cruzada de todos los modelos, cuando se les aplica los parámetros óptimos del *Random Search* sobre obtenidos sobre el modelo *full features*. Se muestra a continuación la evaluación de todos los modelos para las métricas de *accuracy*, *precision*, *recall* y *F1 score*.

Tabla 11: performance *de modelos con parámetros pre-optimizados por* random search

Algoritmo	*Features*	*Accuracy*	*Precision*	*Recall*	*F1 score*
Random Forest	*context*	0.8049	0.8060	0.9979	0.8917
	conversational + context	0.8057	0.8071	0.9972	0.8921
	conversational	0.7991	0.8028	0.9930	0.8878
	conversational + semantic	0.8027	0.8030	0.9987	0.8902
	semantic + context	0.8058	0.8064	0.9985	0.8922
	semantic	0.8016	0.8021	0.9982	0.8895
	conversational + semantic + context	0.8065	0.8069	0.9990	0.8927
XGBoost	*context*	0.8050	0.8050	1	0.8920
	conversational + context	0.8058	0.8058	1	0.8925
	conversational	0.8005	0.8005	1	0.8892
	conversational + semantic	0.8047	0.8054	0.9970	0.8910
	semantic + context	0.8077	0.8080	0.9983	0.8931
	semantic	0.8040	0.8046	0.9971	0.8906
	conversational + semantic + context	0.8082	0.8086	0.9981	0.8935
	Rango (máximo – mínimo)	**0.0074**	**0.0066**	**0.0070**	**0.0057**

Como puede observarse, el mejor modelo por la métrica de evaluación *F1 score* resulta el modelo con las 3 categorías de *features* activos, tanto para el algoritmo *Random Forest* como para el algoritmo *XGBoost*, con un *F1 score* de 0.8927 y 0.8935 respectivamente. Estos dos modelos son los elegidos para realizar el *fine tuning* de hiperparámetros.

Algunas observaciones interesantes:

- No se ven grandes diferencias en cada métrica a través de los distintos modelos: los rangos entre el máximo y el mínimo para cada métrica recién tienen significancia al tercer lugar decimal, como puede observarse en la última fila de la tabla
- El modelo que tiene sólo *features* conversacionales es el que tiene peor *performance* es más cantidad de métricas, tanto para *Random Forest* como para *XGBoost*.
- El modelo *full features* es el que tiene la mejor *performance* es más métricas, incluido el *F1 Score*.

4.3 Resultados de *fine tuning*

Se continúa ahora con el resultado del *fine tuning* para observar los resultados finales con los hiperparámetros optimizados en el conjunto de *train*.

Cabe recordar, que el *fine tuning* se realizó sobre los modelos con mejor *F1 Score*. La *performance* de los modelos con los hiperparámetros de *fine tuning* óptimos ya presentados en la sección anterior, se muestran en la siguiente tabla:

Tabla 12: performance *de modelos con parámetros optimizados por* grid search

Algoritmo	*Features*	*Accuracy*	*Precision*	*Recall*	*F1 score*
Random Forest	*conversational + semantic + context*	0.8076	0.8076	0.9993	0.8933
XGBoost	*conversational + semantic + context*	0.8116	0.8147	0.9919	**0.8946**

Como se observa en la tabla anterior, con los hiperparámetros optimizados por *fine tuning*, el modelo que mejor *performance* tiene por el criterio de *F1 score*, termina siendo el **modelo con algoritmo *XGBoost* y todos los *features***. Por lo tanto, este será el modelo elegido para la evaluación en datos de *test*.

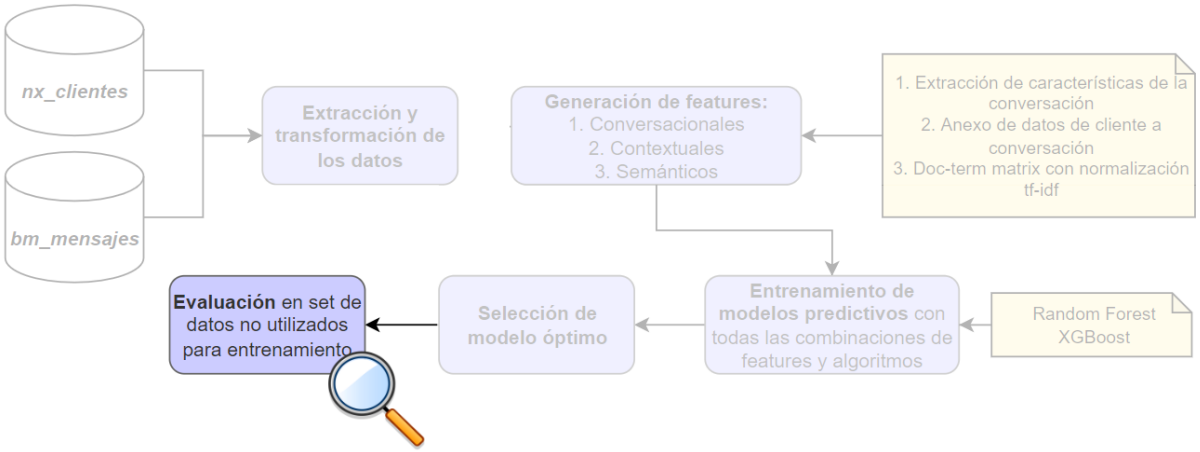
Un punto muy relevante para destacar y ya anticipar los resultados que podríamos ver en la evaluación en datos de *test*: el *accuracy* que muestra el modelo es de 81,2%. Cabe recordar que la proporción de promotores en los datos de entrenamiento es del 80,0%. Con este número puede verse fácilmente que el modelo tiene un *accuracy* sólo 1,2 puntos porcentuales mayor que un modelo que prediga a todos las observaciones como 1's (promotores).

Se evaluarán los datos de *test* para confirmar que el valor predictivo extraído es bajo y resulta importante entender el porqué y posibles mejoras, lo cual se discutirá en la sección de Conclusiones en la página 55.

4.4 Evaluación en datos de *test*

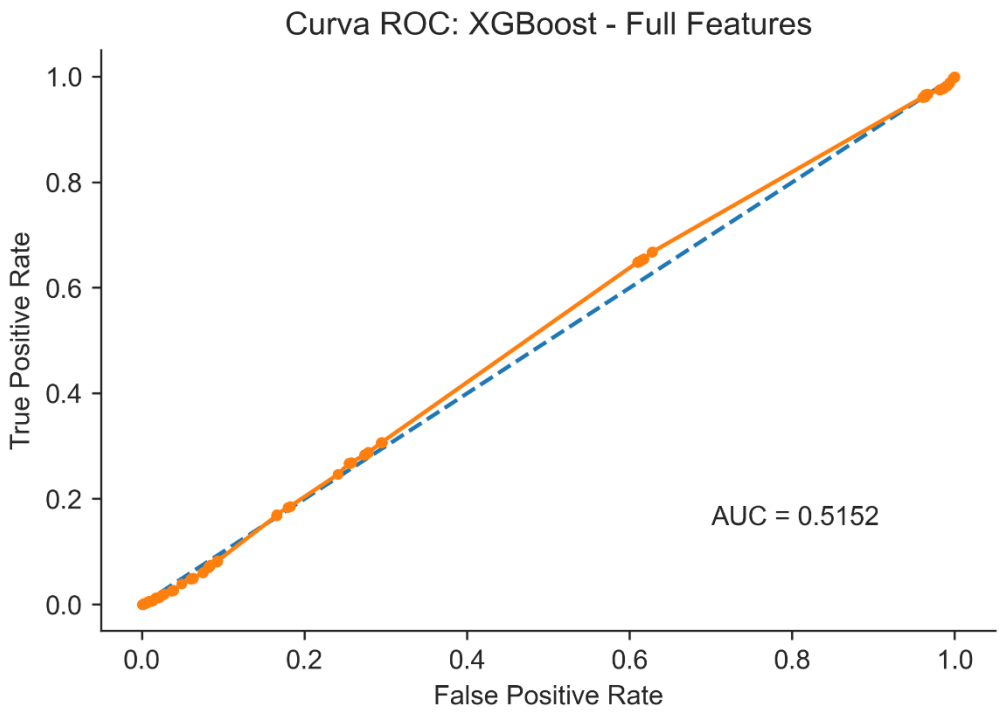
En esta sección se realiza la evaluación del modelo elegido y optimizado sobre los datos de *test* para evaluar la *performance* predictiva del modelo.

La evaluación se realizará sobre el modelo *XGBoost* con todos los *features*, ya que es el modelo que mostró la mejor *performance* en cuanto al *F1 Score*.



Como primer resultado se muestra la curva ROC del modelo:

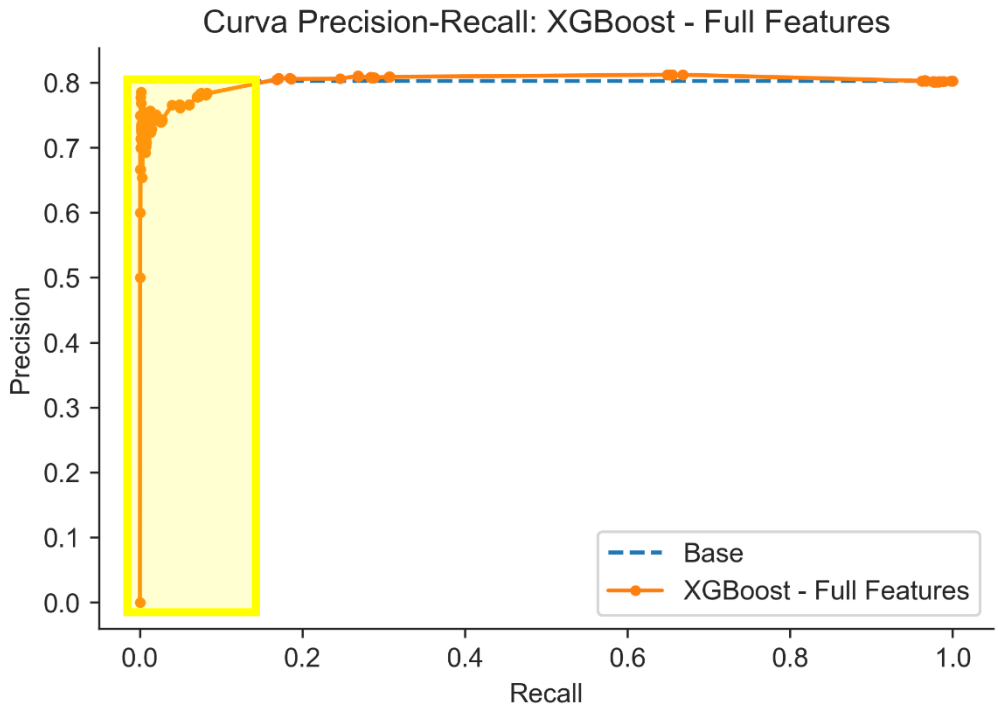
Figura 21: curva ROC del modelo Full Features *con algoritmo* XGBoost *(AUC = 0.5152)*



Como puede observarse en el gráfico, el poder predictivo vs. un modelo sin habilidades aprendidas es apenas incremental, con un AUC de 0,5152.

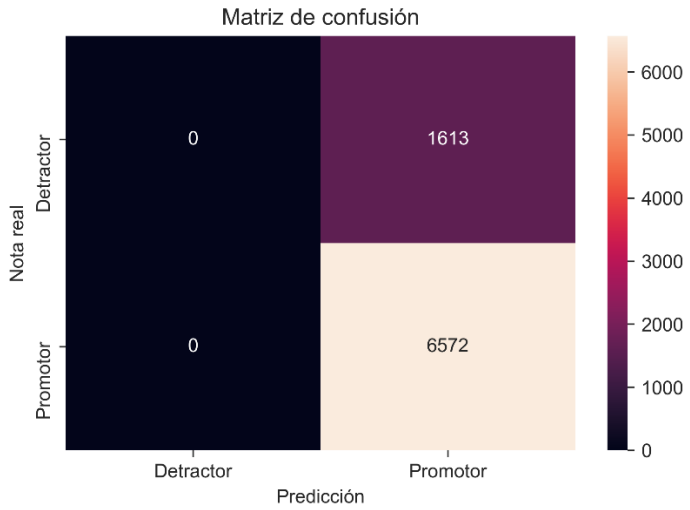
Para complementar este resultado, se presenta la *precision-recall curve* para los datos:

Figura 22: curva precision-recall *del modelo* Full Features *con algoritmo* XGBoost



Para observar los resultados que arrojó la predicción con el *threshold* donde el *F1 score* se maximiza, se presenta la matriz de confusión de dicha predicción:

Figura 23: matriz de confusión para el modelo Full Features *con algoritmo* XGBoost



Por último, se presenta el detalle de los resultados de *scoring* y componentes de la matriz de confusión para todo el rango de *thresholds*, con intervalos resumidos:

Tabla 13: scoring *para el rango total de* thresholds *y detalle de TN, FP, FN y TP para la evaluación en test de* XGBoost Full Features[15]

Threshold	*Accuracy*	*Precision*	*Recall*	*F1 score*	TN	FP	FN	TP
0,6890	0,8029	0,8029	1,0000	**0,8907**	0	1613	0	6572
0,6917	0,8028	0,8030	0,9997	0,8906	1	1612	2	6570
0,6944	0,8022	0,8029	0,9989	0,8902	1	1612	7	6565
0,6971	0,8022	0,8029	0,9989	0,8902	1	1612	7	6565
0,6998	0,8021	0,8028	0,9988	0,8902	1	1612	8	6564
0,7024	0,8009	0,8028	0,9968	0,8894	4	1609	21	6551
0,7051	0,7946	0,8021	0,9880	0,8854	11	1602	79	6493
0,7078	0,7945	0,8021	0,9878	0,8853	11	1602	80	6492
0,7105	0,7908	0,8017	0,9825	0,8829	16	1597	115	6457
0,7132	0,7870	0,8018	0,9761	0,8804	27	1586	157	6415
0,7159	0,7819	0,8032	0,9647	0,8766	60	1553	232	6340
0,7186	0,6010	0,8121	0,6544	0,7248	618	995	2271	4301
0,7213	0,3850	0,8091	0,3063	0,4444	1138	475	4559	2013
0,7240	0,3621	0,8109	0,2681	0,4030	1202	411	4810	1762
0,7267	0,3005	0,8062	0,1697	0,2803	1345	268	5457	1115
0,7293	0,2398	0,7832	0,0736	0,1346	1479	134	6088	484
0,7320	0,2309	0,7658	0,0607	0,1125	1491	122	6173	399
0,7347	0,2242	0,7668	0,0485	0,0913	1516	97	6253	319
0,7374	0,2111	0,7426	0,0268	0,0517	1552	61	6396	176
0,7401	0,2043	0,7287	0,0143	0,0281	1578	35	6478	94
0,7428	0,2038	0,7350	0,0131	0,0257	1582	31	6486	86
0,7455	0,2007	0,7143	0,0076	0,0151	1593	20	6522	50
0,7482	0,1998	0,7391	0,0052	0,0103	1601	12	6538	34
0,7509	0,1982	0,7143	0,0023	0,0046	1607	6	6557	15
0,7536	0,1980	0,7857	0,0017	0,0033	1610	3	6561	11
0,7562	0,1977	0,7778	0,0011	0,0021	1611	2	6565	7
0,7589	0,1974	0,7143	0,0008	0,0015	1611	2	6567	5
0,7616	0,1971	0,5000	0,0002	0,0003	1612	1	6571	1
0,7643	0,1969	-	-		1612	1	6572	0
0,7670	0,1969	-	-		1612	1	6572	0

De los 2 gráficos y la tabla anterior pueden verse claramente los resultados predictivos del modelo:

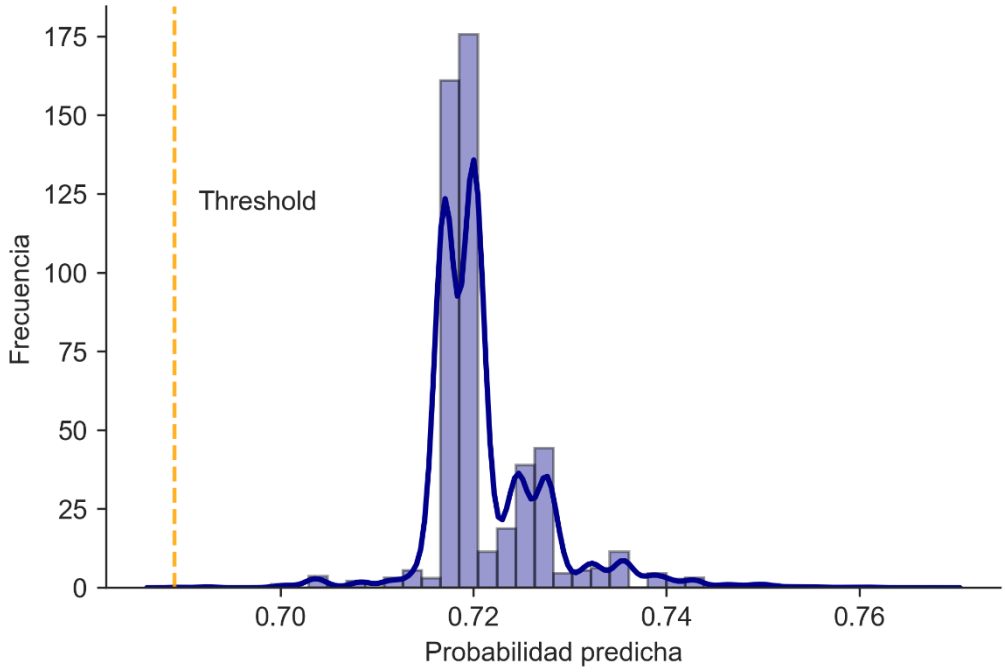
- La curva ROC muestra un área bajo la curva (AUC) muy baja, cercana al 0.5. En los *thresholds* de los extremos se dan valores de *False Positive Rate* y *True Positive Rate* que incluso se encuentran por debajo de la curva.
- la curva de *precision-recall* tiene una forma "atípica": se mantiene en varios puntos por debajo de un modelo "sin poder predictivo" (i.e.: modelo base) debido a que en la parte izquierda del gráfico, señalada en amarillo, hay una volatilidad y una caída grande de la *precision* debido las observaciones sobre las cuáles se hace el cálculo son muy pequeñas (TP + FP) y el modelo

[15] Los *thresholds* mostrados en la tabla no se corresponden exactamente con los del gráfico. Se muestra un rango de *thresholds* sintético reducido para ayudar a la visualización sin perder poder deductivo en las conclusiones extraídas de la tabla.

resulta muy poco exigente, dejando "entrar" a muchos falsos positivos relativo a todas las clasificaciones positivas,

- la maximización del *F1 Score* se da en el *threshold* más bajo con un modelo que predice todas las observaciones como promotores, llegando a un *accuracy* igual a la proporción de promotores en el conjunto de *test*;
- a medida que el modelo se hace más estricto (i.e.: sube el *threshold*), cae rápidamente el *recall* por el rápido aumento de los falsos negativos (i.e.: comienzan a clasificarse como detractores conversaciones que son de promotores)
- Los rangos de *thresholds* van entre 0,6890 (todas predicciones positivas) y 0,7670 (todas predicciones negativas) por lo que pareciera ser un rango muy acotado de las probabilidades predichas por el modelo. A continuación, se muestra el gráfico de distribución de las probabilidades predichas:

Figura 24: distribución de las probabilidades predichas por el clasificador XGBoost Full Features



Puede verse gráficamente que esta optimización del modelo resulta en un modelo cuyo *threshold* óptimo que maximiza el *F1 score* queda por debajo de toda probabilidad predicha.

Para entender qué está sucediendo dentro de esta caja negra que resultó en un modelo óptimo que clasifica a todas las observaciones como positivas. Se detalla a continuación el ranking de *feature importance* de los principales *features*:

Tabla 14: feature importance

Feature	Ganancia total	Ganancia total (% sobre máxima ganancia)
cl_gracias	2.526	100%
cl_muchas	1.065	42%
ag_disculpas	463	18%
msg_per_conv_cliente	351	14%
ag_canal	287	11%
cl_necesito	256	10%
ag_lamento	254	10%
cl_no	234	9%
cl_solucion	229	9%
msg_per_block_cliente	214	8%
ag_pido	204	8%
cl_mushisimas	200	8%
average_delay	184	7%
conv_duration	155	6%

Como puede observarse hay una fuerte importancia en el poder predictivo de las palabras utilizadas por el cliente ("muchas", "gracias", "necesito", "solucion", etc.) con un marcado tono emocional en las palabras.

Asimismo, existe cierta ganancia en las palabras que utiliza el agente ("disculpas", "canal", "lamento", "pido") que entendidas dentro del contexto de uso cobran significado. Usualmente utilizadas en situaciones como las siguientes[16]:

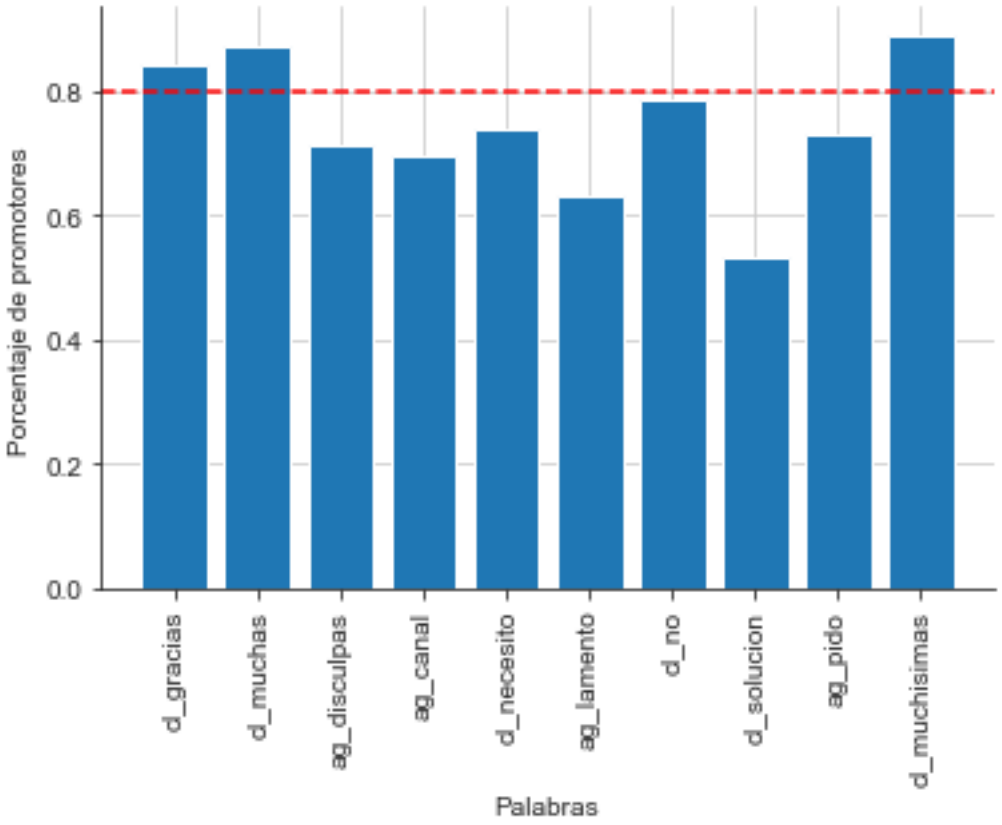
- Cuando un agente queda a la espera de una respuesta de un cliente, y este no responde por más de 7 minutos, los agentes envían un mensaje de la siguiente forma:
  *"Te estuve esperando. Si te quedó alguna consulta nos podés encontrar en este mismo **canal** de lunes a viernes de 6 a 00 hs. ¡Te esperamos! ¡Espero que tengas un hermoso día!"*
- Cuando hay demora en la respuesta del agente, se ven frecuentemente frases como:
  *"Te **pido** **disculpas** por las demoras. Verifico que tu reclamo…"*

Dado que los resultados no mostraron un alto valor predictivo, se mostrarán los resultados a continuación del foco de distribución de promotores sobre los *features* más importantes nombrados en la Tabla 14 para analizar si en estos se encuentra un diferencial en el *mix* de promotores (sugiriendo un mayor poder explicativo) comparado al visto en la Figura 14.

A continuación, se observa el porcentaje de promotores para las conversaciones que tiene presentes las palabras analizadas:

[16] Información de las guías de trabajo de atención al cliente de la empresa, con los scripts propuestos para uso de los agentes

Figura 25: porcentaje de promotores en conversaciones con presencia de distintas palabras



Como se observa en el gráfico, al hacer un *zoom* en las palabras que arrojaron un mayor *feature importance*, sí pareciera haber ahora una relación más marcada entre algunas palabras y el *mix* de conversaciones promotoras (o detractoras, por complementariedad). Al mismo tiempo, al comparar vs. el promedio de *mix* de promotores del conjunto de *test*, puede apreciarse la dirección en la que "empuja" la probabilidad cada una de las palabras. Por ejemplo: cuando los clientes dicen la palabra "solución", sólo el 53% de las conversaciones resultan en promotores (vs. un 80% en el promedio de los datos).

5. Conclusiones

Objetivos propuestos

Vale la pena volver a los objetivos planteados inicialmente en este trabajo para dar una conclusión:

i. Predecir el puntaje con el que un cliente puntuará la atención recibida por un agente a partir de distintas características del cliente, el agente y la conversación
ii. Comprender la importancia para la predicción de las distintas variables disponibles y generadas a partir de las bases disponibles

Con respecto al objetivo i., luego de atravesar el proceso de aprendizaje de conocimiento de los datos y de optimización de un modelo predictivo, el primer objetivo no pareciera estar cumplido al nivel que se esperaba y el segundo sí pareciera estar cumplido, al menos parcialmente.

Lamentablemente, el modelo predictivo optimizado no terminó aportando un gran valor predictivo ya que en el óptimo predijo igual que un modelo sin "*skills*" que predice todos promotores. Parte de la discusión central con este resultado pasa por 3 ejes principales:

a) ¿Qué cuidados se tuvieron en cuenta para asegurar una metodología de trabajo que minimice errores y pérdida de poder predictivo en los modelos?

Existieron varias técnicas y cuidados para asegurar esto. A continuación, se detallan las más importantes:

- En las etapas del proceso en que existió perdida de información (e.g.: *Figura 9: detalle de reducción de información en el procesamiento de las bases*) se aseguró que la perdida no fuese significativa relativa al total de información y que la perdida sea aleatoria, por ejemplo: no existe razón para afirmar que los números de teléfono "perdidos" durante el *join* entre las dos bases de datos, perteneciesen a algún segmento de usuarios con una probabilidad marcadamente mayor/menor que el promedio de usuarios a ser promotores.
- El *data leakage* se evitó tanto para los datos como para la metodología del investigador. Al hacer el *train-test split* al principio del código, se dejó apartado el set de *test* hasta el final. Incluso las transformaciones básicas de los datos se realizaron sobre *train* para evitar cualquier vistazo de los datos de *train* que pudiesen influenciar al investigador. Asimismo, las funciones de *fit*, *encoding* y *scaling* se realizaron siempre sobre *train* y se implementaron funciones que guardaban los parámetros aprendidos en *train* para luego utilizarlos en *test*, sin realizar un *refit*[17].
- Se realizaron distintas versiones (no todas detalladas en la metodología) de la metodología para comparar resultados con la metodología actual y asegurar que no existan diferencias de resultados muy grandes (lo que podría ser signo de una implementación errónea o una elección errónea de las funciones para desarrollar el modelo predictivo). Por ejemplo:
  - ✓ La validación cruzada se probó con distintos métodos (métodos que vienen por default implementados, función *K-Fold* y función *Stratified K-Fold*),
  - ✓ Para descartar problemas de balance de clases se intentó realizar el modelo *XGBoost* balanceado con el factor `scale_pos_weight = 8` y obteniendo resultados muy similares a los resultados sin el escalamiento.
  - ✓ Si bien luego del *fine tuning* ya se obtuvo un modelo ganador, se evaluaron de todas formas los 14 modelos en los datos de *test* para descartar un error de implementación en el código particular del modelo ganador, y se obtuvieron resultados consistentes con lo que se observó en datos de *train*.

[17] Los únicos casos en los que se hizo un *refit* fueron aquellos casos para los cuales los datos categóricos de test contaban con nuevos valores lo que no permitía codificarlos sin el aprendizaje de nuevos valores.

- ✓ Se probaron métricas de *scoring* alternativas como *G-Means*, obteniendo resultados similares.
- ✓ A lo largo del proceso siempre se aplicaron constantemente "verificaciones vs. la realidad". Siempre se le buscó una coherencia con el conocimiento del dominio, por ser parte del trabajo diario, para no confiar únicamente en los datos en la "caja negra", sino que en todos los pasos tengan una interpretabilidad lógica y que tenga sentido. Cada objeto nuevo que se creó en el entorno fue visto varias veces por el ojo del investigador "a mano", sin utilizar ninguna función de resumen de los datos, valiéndose de exportaciones a Excel para visualizar los datos crudos. Así se descubrieron algunos errores como la falta de espacios entre la última palabra de un mensaje y la primera de otro en las funciones de concatenado de mensajes.

b) ¿Qué aprendizajes trajo este resultado acerca de la metodología utilizada?

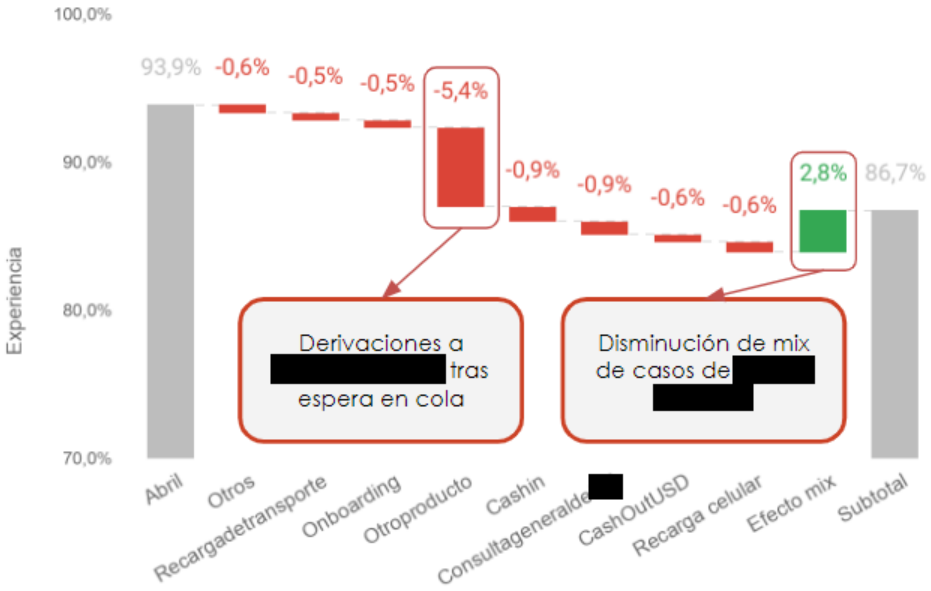
Principalmente dos grandes aprendizajes:

- ✓ **No subestimar la complejidad de la transformación de los datos**. El tiempo dedicado a la transformación de los datos fue superior al 80% del tiempo dedicado y las bases de datos con las que se contó resultaron ser muy difíciles de manejar. Contenían variaciones aleatorias en el orden de los datos desde la fuente de origen, variables embebidas en el medio de textos difícilmente identificables y fue necesario valerse de varias heurísticas que precisaban de adaptaciones para contemplar diversos casos particulares, de forma tal de contemplar todos (o la mayoría) de los casos que buscaba cubrir la función.
- ✓ **Avanzar a la etapa de evaluación más temprano**. La evaluación por validación cruzada, antes de avanzar a *test*, permite tener una orientación rápida, sin necesidad de enormes análisis, acerca de la potencial *performance* de un modelo. Avanzar rápidamente, al menos con algunos parámetros más o menos estándar y con algunos *features* aunque no estén completos, hubiera permitido detectar la baja *performance* del modelo más temprano y dar más tiempo para refinar el *feature engineering* o incluso orientar la estrategia del modelo hacia otro lado.

c) ¿Qué faltó para lograr una mejor predicción?

Pareciera que, además de alguna mejora al modelo, hay algo de información de origen que está faltando para predecir la valoración de la experiencia de atención por parte de los clientes. Observando los reportes de la empresa se observa que hay distintos factores externos que parecen afectar fuertemente la nota de la atención, que podríamos atribuir a "factores de contingencia". A modo de ejemplo, se presenta (con algunos datos ofuscados por cuestiones de confidencialidad) el análisis de evolución de la experiencia de atención al cliente del sector de calidad de la empresa, para los meses de abril y mayo estudiados (en el eje Y se observa la misma variable que se estudió en este trabajo, sólo que normalizada en una escala del 0% al 100%):

Figura 26: gráfico de cascada explicando los componentes de la caída en la variable dependiente de abril a mayo[18]



En el eje X del gráfico se observan lo que se llaman las diferentes tipologías de los chats: cada vez que un agente concluye un chat, asigna una tipología que hace referencia a los motivos de la consulta o reclamo. Además, se muestro un efecto *mix*, que hace referencia a las variaciones en la métrica *month over month* a causa de variaciones en el *mix* de tipologías.

Como se observa, hay claras variaciones en los niveles de la variable causados por contingencias puntuales asociadas a alguna tipología en particular que no tienen tanto que ver con los 3 grupos de *features* utilizados en este trabajo, sino más bien con el tema se está consultando (que a veces no es capturado del todo por las variables semánticas planteadas en este trabajo).

A modo de ejemplo, en la Figura 26, lo que sucedió en abril fue que se disponibilizaron en comunicaciones a clientes los *links* para acceder a la atención al cliente para WhatsApp, para clientes de otro tipo de productos, distintos al *expertise* de los agentes. Al tener mayor *incoming* de chats, se saturó la capacidad y se elevaron los tiempos de espera. La experiencia de la gente que quería consultar por este otro producto cayó en particular mucho, ya que no sólo tenían un alto tiempo de espera como el resto de las tipologías, sino que encima, al final de esa espera no había resolución, ya que se los derivaba con otros agentes que sí tuviesen el *expertise* en el producto.

Más allá del ejemplo puntual, puede verse (y además se corrobora en conversación con gente de la empresa) que hay un gran peso de factores "externos" o de "contingencia" que hacen a las variaciones en el puntaje que se da a la conversación.

Con respecto al objetivo ii., resultó haber un aprendizaje sobre la importancia de las variables o *features* elegidos y es que no resultaron tener un gran poder predictivo para la predicción de la variable dependiente. En el análisis del funcionamiento del modelo predictivo, se logró detectar alguna palabra que, al transformar sus valores en variables binarias (palabra presente en la conversación = 1, palabra ausente en la conversación = 0), pareció tener un poder de clasificación mayor, ya que su presencia reduce el número de promotores de forma bastante significativa: de 80% a 53% (ver Figura 25). Claro que esta reducción al 53% nos indica que sería un clasificador lejos de perfecto, pero al menos se descubrió alguna correlación.

[18] Informe de calidad del mes de mayo del sector de Customer Experience de la empresa

Por otro lado, de la Tabla 14 surge una visión de las variables generales que, si bien con poder explicativo, resultan ser las más importantes en cuanto a las características semánticas y conversacionales: el agradecimiento de los clientes es una señal fuerte, como es de esperar, de una nota promotora.

Como se sabe, igualmente, esto no resulta en un poder predictivo alto. Es por eso que a continuación se detallan algunas oportunidades de mejora para el modelo predictivo.

Oportunidades de mejora

A continuación, se enumeran algunas oportunidades de mejora encontradas para el modelo predictivo, algunas ya anticipadas más arriba, con el objetivo de guiar cambios en el proceso de una futura investigación sobre este tema:

1. **Profundizar el *feature engineering***. Buscar nuevas variables con los *features* que puedan tener mayor valor explicativo, como ser la binarización de la presencia de palabras en conversaciones (vs. normalización tf-idf)
2. **Añadir más fuentes de datos**. En este sentido se propone la adición de datos que no pudieron conseguirse de las bases disponibles como información sobre la transaccionalidad de los clientes en la aplicación, pero más importante, la tipificación de las temáticas de los chats (e.g.: teniendo columnas de 0's y 1's que indique la tipificación que resultó en cada conversación).
3. **Incorporación de datos de contingencias del producto**. Como se vio en la Figura 26, la información de eventos externos de error o de contingencias de producto (no disponible en las bases actuales) pareciera ser bastante relevante y podría ser muy útil tener la información de, por ejemplo, qué días se presentó una contingencia del producto (i.e.: un error en alguna funcionalidad).
4. **Incorporación de *word embeddings*** (por ejemplo: Word2Vec o Bert)[19]. Posiblemente, con el uso de *word embeddings*, se podría haber ayudado a la interpretación y quizás el *binning* de algunos *features* semánticos. Encontrando niveles altos de similaridad entre palabras, se podrían haber "juntado" grupos de *features* semánticos (como "cl_muchas" y "cl_gracias") para ayudar en la construcción de *features* más sólidos que probablemente, si están separados, están positivamente correlacionados entre sí.

Aportes de valor

El principal aporte de valor de este trabajo fue poder dar luz sobre el bajo poder predictivo de las características conversacionales, contextuales y semánticas de las conversaciones de WhatsApp, para la predicción del puntaje otorgado a los agentes por parte de los clientes.

Se lograron entender algunos de los componentes que sí aportan algo de valor predictivo, pero, sobre todo, se logró determinar la baja importancia de los factores utilizados para la predicción de la nota.

Esto puede ser de valor para la empresa para enfocar futuras investigaciones orientadas a las oportunidades de mejora propuestas, por ejemplo, con nuevas fuentes de datos que, de confirmarse, podrían encaminar el enfoque de los esfuerzos en aquellas variables que sí tengan mayor poder explicativo.

---

[19] Una idea similar se sugiere en Mohammad, K. S., & Hemmatian, F. (2019). *An efficient preprocessing method for supervised sentiment analysis by converting sentences to numerical vectors: A twitter case study. Multimedia Tools and Applications*.

Resultaría interesante conectar futuros modelos predictivos con variables que sean posteriores en el ciclo de vida del cliente, como puede ser el *churn* de los usuarios, para poder predecir la pérdida de un usuario a partir de lo sucedido en un contacto con atención al cliente.

6. Referencias

- Fred Reichheld, *La Pregunta Decisiva 2.0*, LID Editorial, 2012
- Freed Larry, *Innovating Analytics*, John Wiley & Sons Inc., 2013
- Mohammad, K. S., & Hemmatian, F. (2019). *An efficient preprocessing method for supervised sentiment analysis by converting sentences to numerical vectors: A twitter case study.* Multimedia Tools and Applications, 78(17), 24863-24882
- Gálvez Ramiro, Gauder Lara, Luque Jordi & Gravano Agustín, *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, p. 215-224, Association for Computational Linguistics, 2020
- *A guide to XGBoost hyperparameters* (https://towardsdatascience.com/a-guide-to-xgboost-hyperparameters-87980c7f44a9)
- *Fine-tuning XGBoost in Python like a boss* (https://towardsdatascience.com/fine-tuning-xgboost-in-python-like-a-boss-b4543ed8b1e)
- *Hyperparameter Tuning For XGBoost: Grid Search Vs Random Search Vs Bayesian Optimization* (https://grabngoinfo.com/hyperparameter-tuning-for-xgboost-grid-search-vs-random-search-vs-bayesian-optimization)
- *Performance Metrics: Confusion matrix, Precision, Recall, and F1 Score* (https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262)
- *A Gentle Introduction to XGBoost Loss Functions* (https://machinelearningmastery.com/xgboost-loss-functions)
- *How to Tune the Number and Size of Decision Trees with XGBoost in Python* (https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python)
- *Feature Importance and Feature Selection With XGBoost in Python* (https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python)
- *Xgboost Feature Importance Computed in 3 Ways with Python* (https://mljar.com/blog/feature-importance-xgboost)
- *Understanding Random Forests Classifiers in Python Tutorial* (https://www.datacamp.com/tutorial/random-forests-classifier-python)
- *Evaluating a Random Forest model* (https://medium.com/analytics-vidhya/evaluating-a-random-forest-model-9d165595ad56)
- *Hyperparameter Tuning the Random Forest in Python* (https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74)
- *A Gentle Introduction to Threshold-Moving for Imbalanced Classification* (https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification)

Apéndice A: Descripción de los datos de origen

Base de datos	Nombre columna	Descripción
bm_mensajes	*id*	Identificador único de cada línea/mensaje.
bm_mensajes	*de*	Indicador del emisor del mensaje. Para mensajes enviados por el *bot* tiene el valor "Bot", para mensajes enviados por el cliente muestra el alias de WhatsApp (e.g.: Santi 👌😉) y para mensajes enviados por los agentes muestra el nombre y apellido del agente.
bm_mensajes	*id_contacto*	Teléfono celular desde el cuál se contacta el cliente.
bm_mensajes	*fecha_creacion*	Fecha de envío del mensaje
bm_mensajes	*operador*	Indicador que muestra el nombre y apellido del agente que envía el mensaje. Vacío si el mensaje es enviado por un cliente o el *bot*.
bm_mensajes	*mensaje*	Campo de tipo *string* conteniendo el texto del mensaje enviado por el emisor. Cuando el mensaje enviado resulta ser la elección de un botón/opción de un menú, en el texto se anexan algunos *tags* identificatorios de ese evento (e.g.: {"button":"Ingreso a la app"} es un *tag* del campo mensaje que indica la elección del cliente de la opción "Ingreso a la app" dentro del menú de opciones que el *bot* le ofreció)
nx_clientes	*id_global*	Identificador de cliente único generado por la empresa al momento del alta.
nx_clientes	*nacionalidad*	Nacionalidad declarada por el cliente al momento del registro.
nx_clientes	*provincia*	Provincia declarada por el cliente al momento del registro.
nx_clientes	*código_postal*	Código postal declarado por el cliente al momento del registro.
nx_clientes	*teléfono*	Número de teléfono indicado por el usuario al momento de registrarse en la aplicación. Este dato fue obligatorio en algunos momentos y no obligatorio en otros al momento del registro.
nx_clientes	*sexo*	Toma los valores M (masculino) o F (femenino)
nx_clientes	*fiscal_position*	Indicador de la condición fiscal (e.g.: monotribusita, consumidor final, responsable inscripto, etc.) declarada por el cliente al momento del registro.
nx_clientes	*estado_civil*	Indica el estado civil declarado por el cliente al momento del registro.

Apéndice B: *Unit testing*

Se realizó *unit testing* para 24 funciones del script *script_mensajes.py* para asegurar la calidad de las funciones en el remanejo de la información cruda en su camino hacia el *dataframe* final que se utiliza en el script *modelos.py* para entrenar los modelos predictivos.

A continuación, la descripción de los *tests* que se pueden encontrar en *unit_testing.py*:

Función	¿Qué se quiere *testear*?	Criterios del *unit test*	Resultado
train_test_subset	Completitud	1. Largo *bm_mensajes* = largo *train* + largo *test* 2. Número de mensajes en train_data_features = 70% (±2%) Número de mensajes originales[20] 3. Largo matriz *train_test_subset* = largo train + largo *test*	1. Exitoso 2. Exitoso 3. Exitoso
train_test_split	Data Leakage	1. No existencia de clientes de *train* en *test* 2. No existencia de clientes de *test* en *train*	1. Exitoso 2. Exitoso
tag_button_no	Ejecución de función activa	Número de mensajes en *tag_button_no* con etiqueta "button_no" > 0	Exitoso
tag_button_si	Ejecución de función activa	Número de mensajes en *tag_button_si* con etiqueta "button_si" > 0	Exitoso
tag_wait_offer	Ejecución de función activa	Número de mensajes en *tag_wait_offer* con etiqueta "wait_offer" > 0	Exitoso
tag_agent_talk	Ejecución de función activa	Número de mensajes en *tag_agent_talk* con etiqueta "agent_talk" > 0	Exitoso
tag_derivation	Ejecución de función activa	Número de mensajes en *tag_derivation* con etiqueta "derivation" > 0	Exitoso
tag_day_end	Ejecución de función activa	Número de mensajes en *tag_day_end* con etiqueta "day_end" > 0	Exitoso
tag_no_more_wait	Ejecución de función activa	Número de mensajes en *tag_no_more_wait* con etiqueta "no_more_wait" > 0	Exitoso
tag_agent_close	Ejecución de función activa	Número de mensajes en *tag_agent_close* con etiqueta "agent_close" > 0	Exitoso
tag_conv_evet	Ejecución de función activa	Número de mensajes en *conv_event* con etiqueta "agent_close" ó "no_more_wait" ó "day_end" ó "derivation" > 0	Exitoso
assign_conv_id	Ejecución completa Lugar de ejecución correcto en el *dataframe*	1. *conv_id* máximo = número de *conv_id* distintos 2. Línea con evento *derivation* tiene *conv_id* asignada	1. Exitoso 2. Exitoso
sequence_conversations	Lugar de ejecución correcto en el *dataframe*	Número de líneas con *conv_seq* = 1 es igual a número de *conv_id's*	Exitoso
assign_block_id	Completitud	Número de *block_id* con números enteros = número de mensajes	Exitoso

[20] Se considera un margen de error de 2% ya que al hacer el split en train y test, la separación no se hace a nivel mensajes (lo que daría un 70% exacto) sino que el split se hace a nivel clientes, por lo que el número exacto puede variar levemente, dado que no todos los clientes tienen asignada la misma cantidad de mensajes.

Función	¿Qué se quiere testear?	Criterios del *unit test*	Resultado
extract_agent_score	Completitud No duplicidad	1. Número de datos completos en *score_agent* = número de datos completos en *score_agent_conv_id* 2. Datos únicos en *score_agent_conv_id* = número de datos completos en *score_agent_conv_id*	Exitoso
extract_conv_duration	Completitud	Número de datos completos en *conv_duration* = número de datos completos en *score_agent_conv_id*	Exitoso
image_sent_tag	Completitud	Número de datos completos en *image_sent* = número de datos completos en *score_agent_conv_id*	Exitoso
*wait_offer_count**	Completitud	Número de datos completos en *wait_offer_count* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
msg_per_conv_agente	Completitud	Número de datos completos en *msg_per_conv_agente* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
*msg_per_conv_cliente**	Completitud	Número de datos completos en *msg_per_conv_cliente* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
*msg_per_block_agente**	Completitud	Número de datos completos en *msg_per_block_agente* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
*msg_per_block_cliente**	Completitud	Número de datos completos en *msg_per_block_cliente* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
*msg_per_conv_total**	Completitud	Número de datos completos en *msg_per_conv_total* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
*agent_delay**	Completitud	Número de datos completos en *averege_dealy* y *max_delay* > 99% número de datos completos en *score_agent_conv_id*	Exitoso
tag_weekday	Completitud	Número de datos completos en *weekday* = número de datos completos en *score_agent_conv_id*	Exitoso

*Para las funciones marcadas con asterisco, se testeó la completitud de los datos al 99%. El motivo de esto es que las funciones se implementaron asumiendo un funcionamiento correcto de la plataforma sobre la cual se envía la mensajería, y sin ningún *bug* o *testing* de formas de operación de los chats que pueda haber hecho la compañía. Sin embargo, se obtuvieron resultados que no son del 100%, y oscilaron entre 99,48% y 99,58% de completitud. Al revisar los datos de origen, se vio que esto fue causado por algunas conversaciones cuya estructura de datos resultaba anormal (e.g.: inicia una conversación sin que el *bot* avise al cliente que se está derivando a un asesor, lo que hace que no se identifique toda esa conversación como tal, ya que no tiene un evento de inicio).

La decisión, dados estos casos fue tolerar hasta un 1% de estos datos incompletos, entendiendo que este umbral tan bajo no debería alterar los resultados predictivos de manera significativa.