Telecom: Predicción de baja de clientes mediante una red neuronal como meta-modelo

Loncón Joaquín

31 de mayo 2022

Contenido

Introducción

La baja de los clientes representa un problema importante para las empresas de telecomunicaciones ya que es mas costoso conseguir nuevos clientes en comparación a mantener los que ya tienen. Por lo tanto, las empresas prefieren evitar que sus clientes se den baja pasándose a la competencia.

Objetivo

Para abordar este proyecto nos pondremos en el lugar de una de estas empresas de telecomunicaciones. Nuestro objetivo es **predecir si un cliente se va a dar de baja de nuestros servicios o se mantendrá con nosotros**.

Metodología

Abordaremos el problema desde un **enfoque dirigido** (Targeted approach), este enfoque se basa en la identificación de los clientes que probablemente abandonen el servicio (lo cual va en linea con nuestros objetivos) y en una intervención adecuada para animarles a quedarse. Notemos que este enfoque puede suponer grandes pérdidas para la empresa si las predicciones de abandono son inexactas, ya que la empresa estaría desperdiciando el dinero de los incentivos en clientes que se habrían quedado de todos modos.

Con el objetivo de obtener predicciones exactas sobre el abandono de los clientes se terminará entrenando una **Red neuronal** (neural network) la cual se basará en las predicciones de varios modelos de **Machine Learning**. A lo largo del proyecto se detallará como se lleva a cabo esto.

Sumado a esto, al entrenar cualquier modelo se realizará un *pre-processing* (pre-procesamiento de los datos), el cual omitirá variables con varianza cero y además, como la base contiene *prevalence* (predominio) de una clase (en nuestro caso, la mayoría de los clientes se mantienen con nosotros), realizaremos una técnica llamada up-sampling, la cual consiste en incrementar el tamaño de la clase minoritaria mediante un muestreo con reemplazo, es decir, la muestra extraída se repone cada vez para que la población sea idéntica en cada extracción, esto se realiza hasta que ambas clases tengan el mismo tamaño. Por último, los predictores serán normalizados (centrados y escalados) cuando corresponda y se llevará a cabo un *10-fold Cross Validation* para cada modelo.

Datos

Se utilizarán los datos de una empresa de telecomunicaciones (telecom para abreviar). Por razones de confidencialidad, el nombre de la empresa de telecom queda en el anonimato. La base de datos es pública y es provista por la comunidad de CrowdAnalytix.

La variable que buscamos predecir se llama **Churn**, la cual toma el valor 1 cuando el cliente se da de baja de nuestros servicios y 0 cuando continua con nosotros. Para su mejor interpretación renombraremos sus valores como `Churn` y `No_churn` respectivamente.

Algunas de las variables que nos servirán para llevar a cabo las predicciones son:

- International plan: toma el valor 1 si el cliente tiene un plan internacional y 0 en cualquier otro caso.
- Total day calls: representa el total de llamadas realizadas en el día.

Entre otras, para saber más sobre la base y sus variables se puede visitar Kaggle y CrowdAnalytix.

Preparación

En esta sección se deja la base de datos lista para su posterior uso.

Missing values

No se observa ningún valor faltante (*missing value*) en la base de datos, por lo cual podemos continuar.

Duplicados

No se observa ninguna observación duplicada en la base de datos, por lo cual podemos continuar.

Meta-Modelo

Una vez que revisamos que la base esta completa y libre de errores podemos comenzar a construir nuestro modelo.

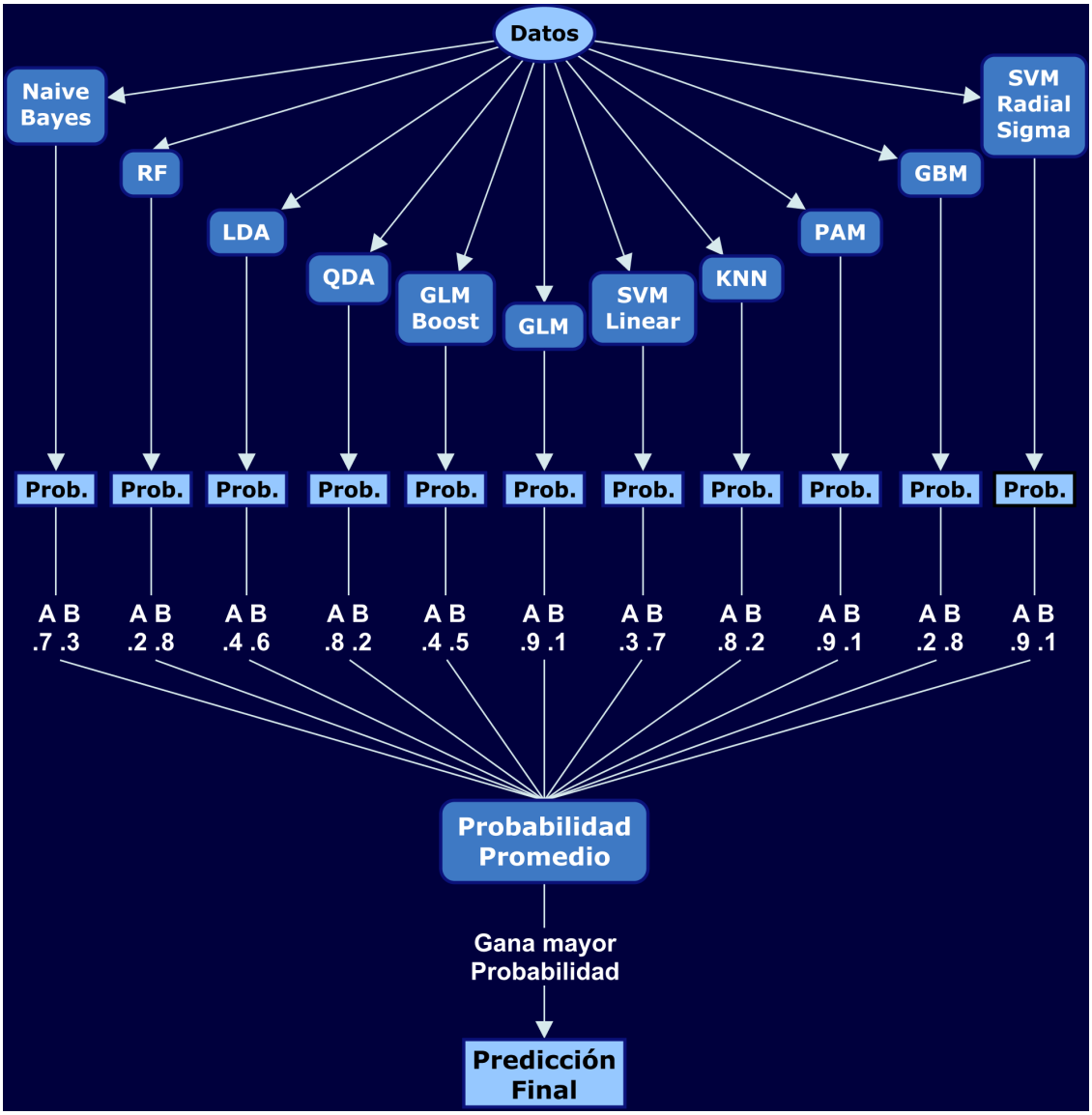
Concepto de Ensemble

Un *Ensemble* es el concepto de combinar los resultados de los diferentes algoritmos para hacer predicciones, es decir, "juntar" muchos algoritmos en uno solo, esto generalmente mejora mucho los resultados finales.

En un problema de clasificación con solo dos categorías, como en el que nos encontramos ahora, podemos encontrar dos maneras de hacer un ensemble.

Promedio de probabilidades condicionales

Cada algoritmo estima una probabilidad condicional para cada posible resultado. Esto es, si tenemos solo dos posibles categorías A o B, para cada observación los algoritmos estiman la probabilidad de que esa observación sea A o B, condicional en los predictores (es decir, la probabilidad dado los valores que toman predictores). Luego se toma la probabilidad de A de todos los algoritmos y se la promedia, lo mismo se hace para la probabilidad de B, obteniendo así una **probabilidad condicional promedio** . Por último, el resultado con mayor probabilidad es el que se predice.

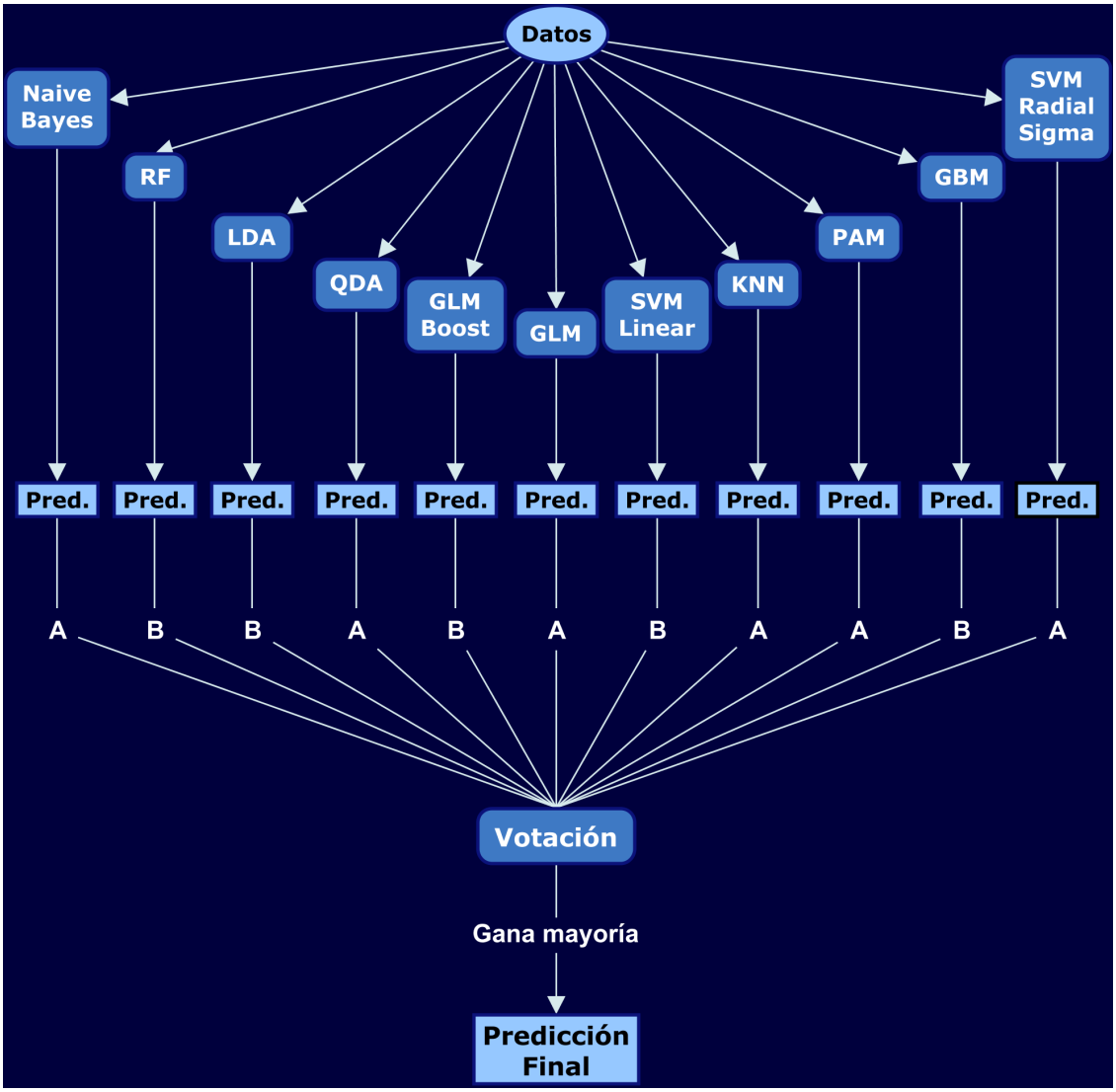


Si quieren ver ejemplos prácticos de este método pueden revisar otros de mis proyectos:

- Ecommerce: Predicción de intención de compra en linea
- Company Bankruptcy Prediction
- Credit Card Fraud Detection
- Credit Card Default Prediction

Votación

Quizás el método mas fácil de interpretar. Como sabemos cada algoritmo predice A o B, lo que se hace en este caso es hacer que los algoritmos "voten" la clase que se predice. Si la mayoría de los algoritmos predice A, entonces la predicción final será A.



El problema que puede surgir de este último método (el primero tampoco está exento de problemas) es que quizás haya algoritmos que como son de la misma familia predicen igual, por lo tanto sería como si le estuviéramos dando 2 votos a una predicción. También puede pasar que, por la naturaleza del problema, la mayoría de los algoritmos prediga mal y al ser mayoría será su predicción la que se obtenga.

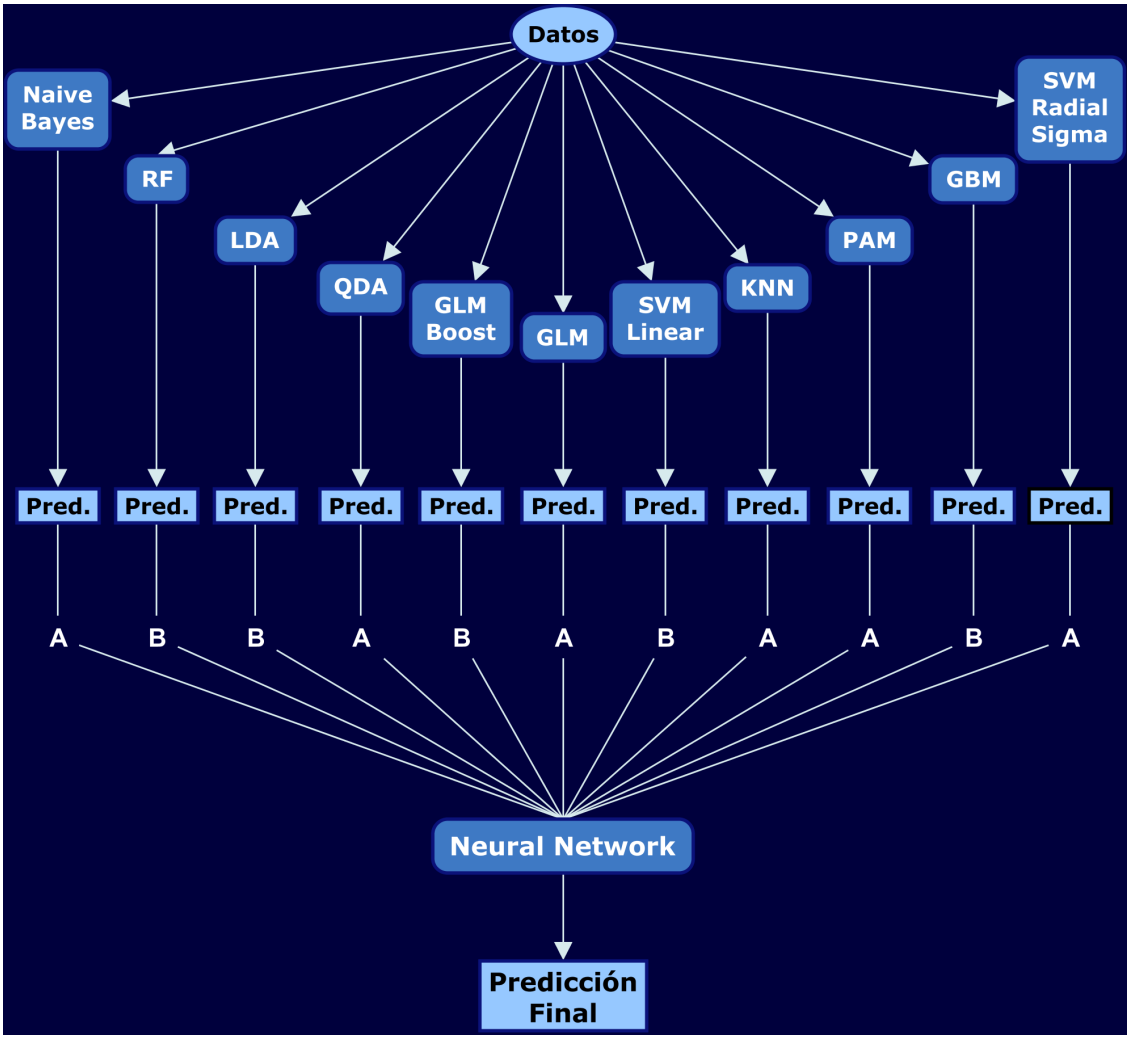
Consecuencia de estos problemas surgen preguntas como: ¿Se pueden seleccionar los mejores algoritmos y descartar los peores? ¿y si ponderamos más (le damos mas votos) a ciertos algoritmos? ¿Como optimizamos este proceso?. Veremos como solucionar estas interrogantes en la próxima sección.

Concepto de Meta-Model

Un meta-modelo (**Meta-model**) es el segundo paso del proceso llamado **Stacking**. Este proceso consiste en entrenar distintos modelos de **Machine Learning**, cuyas predicciones servirán de base para la predicción final de un modelo en particular. El modelo final que predice en base a las demás predicciones se lo denomina **meta-model**. En nuestro caso este meta-model será una **red neuronal** (neural network).

Como vimos anteriormente en la sección de Concepto de Ensemble - votación, existen varios problemas a la hora de decidir como "votan" los modelos. Si lo pensamos como un problema de machine learning, lo que

hacíamos era terminar prediciendo la clase que mas predecían los modelos, esto lo hacíamos con el **fin de obtener una mejor predicción**, pero entonces, ¿como sabemos que esa combinación es la mejor forma? quizás hay modelos que se desempeñan mejor que otros en general o quizás en ciertos casos conviene utilizar unos y a veces otros. Como sería muy difícil para una persona decidir correctamente esto, lo que hacemos es enseñarle a una red neuronal a predecir correctamente utilizando como **input** las predicciones obtenidas de los modelos. De esta forma, una vez entrenada la red neuronal, podrá estipular su predicción final en base a lo que predicen los otros modelos.



Creación del Meta-Model

Una vez entendido como funcionará nuestro meta-model viene la parte de crearlo y ni bien empezamos ya nos encontramos con el primer problema: ¿Cómo entrenamos nuestro meta-model?

Pensemos lo siguiente, los modelos base se entrenan en el **train_set** (80% de los datos) para luego predecir en el **test_set** (20% de los datos), pero entonces ¿donde entrena nuestro meta-model?. Bien, uno podría pensar que se pueden utilizar las predicciones del test_set y entonces entrenar el meta-modelo en ese set, esta opción es válida, cuando se realiza de esta forma sin embargo se denomina **Blending** y no Stacking (blending es la generalización del Stacking). Pero el problema que surge es que se entrena con pocos datos dada la partición (20%) y se necesitaría otro set para poder testear la performance del meta-model.

La opción de entrenar los modelos base en el train_set y hacer las predicciones ahí mismo NO es válida debido al **overfitting** (sobreajuste del modelo a los datos). Sin embargo, esta opción va encaminada a la alternativa que usaremos, ya que terminaremos con predicciones para cada observación del train_set pero sin correr el riesgo del overfitting.

input

Recordemos que el input es simplemente las predicciones de los modelos base. Lo que haremos es tomar el train_set y dividirlo en 10 partes sin solapamiento, a estas partes las llamaremos **folds**, de esta forma cada fold contendrá el 10% de los datos del train_set y con observaciones únicas, es decir, una observación jamás aparecerá en 2 folds distintos (esto es sin solapamiento).

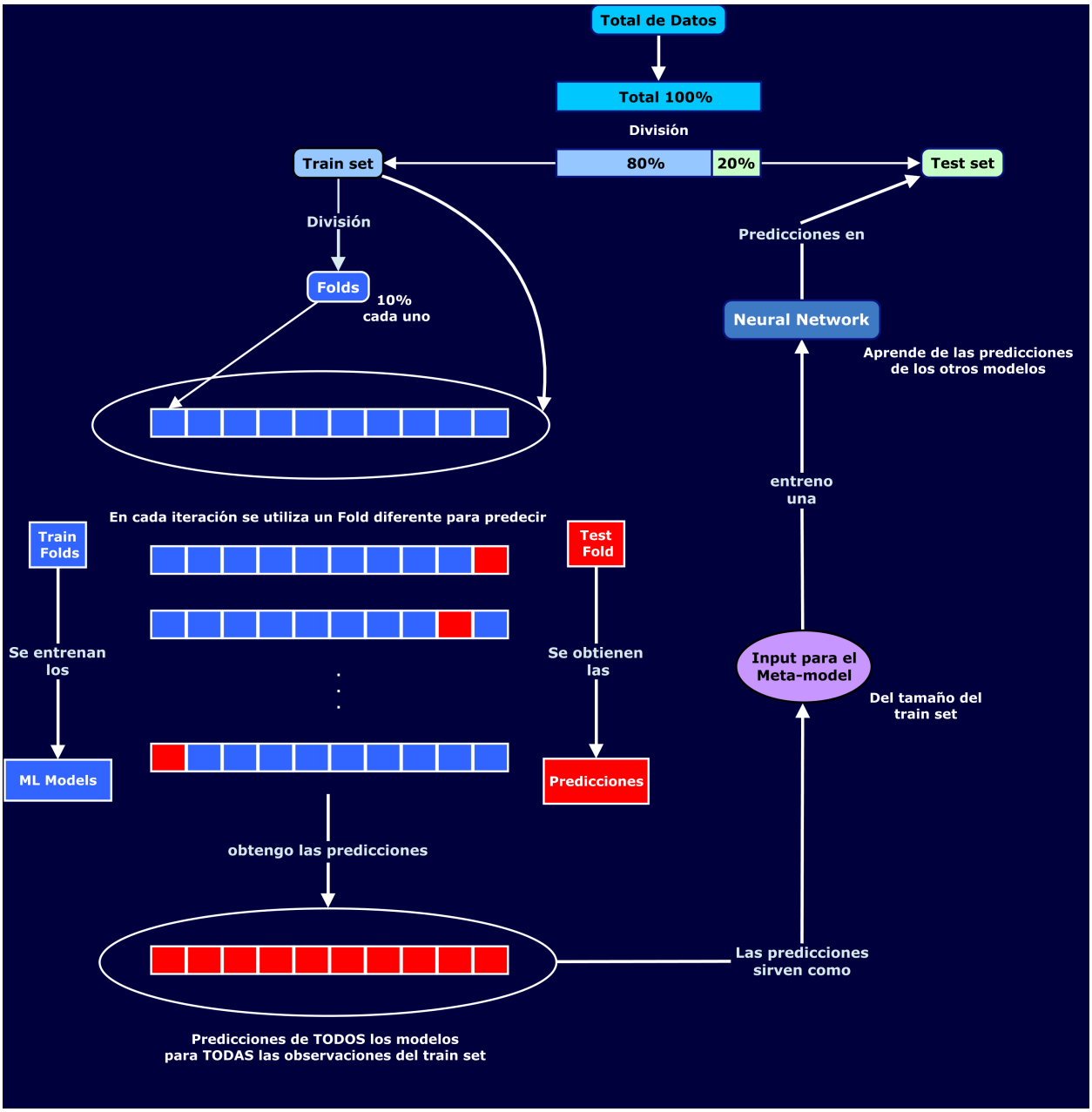
Lo que haremos seguidamente es entrenar los modelos en 9 de los 10 folds y dejar el último fold (10°) para hacer las predicciones. En una segunda instancia utilizamos todos los folds para entrenar excepto el 9°. Luego dejamos el 8°, el 7° y así sucesivamente hasta haber utilizado todos los folds para obtener las predicciones (Nota: este método se llama *k-fold cross validation*, con k = 10). Notemos que de esta forma entrenamos los modelos a lo largo de todo el train_set y también obtuvimos las predicciones para todo train_set evitando el overfitting, ya que en ningún momento entrenamos el modelo e hicimos las predicciones en un mismo fold, sino que siempre fuimos dejando uno separado para predecir.

Al finalizar este procedimiento terminamos con la predicción de todos los modelos para todas las observaciones, esto será el input de nuestro meta-model, el cual tendrá el mismo número de observaciones que el train_set.

Red neuronal

Como dijimos anteriormente, el modelo que elegimos como nuestro meta-model es una red neuronal, esta toma como input las predicciones que se obtuvieron en el train_set mediante *10-fold Cross Validation*. Seguidamente se entrena la red neuronal, siendo la variable a predecir la observación de si un cliente se da de baja de nuestros servicios o no y los predictores las predicciones de los modelos base.

Una vez la red neuronal aprende como utilizar las predicciones de los modelos base para mejorar la predicción final observamos su performance en el test_set, el cual no habíamos utilizado hasta ahora.



Resultados

Performance del Meta-model

Finalmente pasamos a ver como se desempeñó nuestra red neuronal como meta-model.

Recordemos que la precisión es la proporción de las clases que predecimos correctamente, como en nuestro caso teníamos un problema de predominio que corregimos, también veremos la precisión balanceada .

Nuestra precisión (**Accuracy**) es del 92.19%. Pero, por el problema del desbalance de las clases podemos observar nuestra precisión balanceada, que pondera de la misma forma ambas clases, de esta manera si predecimos bien una sola, pero muy mal la otra nuestra precisión balanceada caerá. El valor que toma nuestra precisión balanceada (**Balanced Accuracy**) es de 88.08%.

por otro lado, si queremos saber del total de clientes que se dan de baja de nuestros servicios ¿cuántos detectamos previamente? usamos la métrica llamada **recall**, la cual muestra que los detectamos bien un 82.29% de las veces. Pero cuidado, por como describimos nuestro problema esto no nos sirve para tomar decisiones, ya que podríamos tener un alto nivel de detección de estos clientes, pero también podríamos estar prediciendo erróneamente que aquellos clientes que se quieren quedar con nosotros se van a ir, entonces nuestra empresa estaría desperdiciando el dinero de los incentivos en clientes que se habrían quedado de todos modos.

Por el **enfoque dirigido** (Targeted approach) que adoptamos, debemos revisar nuestra métrica llamada Positive Predicted Value (**PPV**). Esta nos dice que, cuando predecimos que un cliente se quiere dar de baja, el 69.3% de las veces esa predicción es correcta.

Si quisiéramos aumentar nuestra PPV podríamos incluso tomar las probabilidades que genera nuestra red neuronal para cada clase y limitar cuando predice. Esto sería, predecir que el cliente se quiere dar de baja solamente cuando estimamos una alta probabilidad (supongamos un 90%) de que así sea. De esta forma cuando predecimos que un cliente se da de baja, como solo lo hacemos al estar muy seguros, nuestras predicciones de esta clase van a ser mucho mas precisas. Sin embargo, cabe destacar que existe un **trade-off** entre recall y PPV, si queremos aumentar nuestra PPV, es decir, predecir con menos errores, inevitablemente detectaríamos menos a los clientes que nos abandonan, ya que como solo predecimos cuando estamos muy seguros, dejamos pasar muchas observaciones de clientes que efectivamente nos iban a dejar, pero no estábamos tan seguros como para predecirlo.

Por último, observando lo correspondiente a la otra clase que estamos analizando, los clientes que NO nos abandonan y se quedan con nosotros. De las veces que predecimos que un cliente se queda con nosotros, el 96.92% acertamos.

Conclusiones y acciones a tomar

Dado los buenos resultados de nuestra red neuronal, vimos que podemos predecir con bastante precisión si un cliente se va a dar de baja de nuestros servicios o se mantendrá con nosotros y por lo tanto pudimos cumplir nuestro objetivo desde un enfoque dirigido.

Bajo nuestro enfoque, cuando predecimos que los clientes se quieren dar de baja los podríamos incentivar a quedarse con nosotros mediante promociones especiales, un mayor acercamiento a nuestros clientes mediante publicidades específicas y otro sin fin de acciones que podemos tomar como empresa de telecom ahora que tenemos la capacidad de predecir algo tan importante para nuestra empresa. Además, lo podemos hacer con la seguridad de no estar desperdiciando nuestros recursos dada nuestra alta exactitud al momento de predecir.