Reconocimiento de Patrones

Proyecto Final

Lucas Langwagen
Statistical Analyst
PayGroup
The Payment Company

Índice general

Introducción

El presente documento constituye el trabajo final de la materia de Reconocimiento de Patrones y versa sobre la construcción de un modelo neural para detección de fraude en tarjetas de crédito.

Comencé a trabajar en este problema desde el mes de Julio como parte de mis tareas como analista en la empresa PayGroup, es decir que las primeras etapas se llevaron a cabo fuera del contexto del curso. En consecuencia, este documento debe entenderse como una recopilación y análisis crítico del trabajo realizado conjuntamente dentro de la empresa y por fuera como proyecto para esta materia de posgrado. Particularmente, es posible que los procedimientos, decisiones o experimentos relatados no sean los más adecuados ni se condigan con las prácticas recomendadas en el curso. En estos casos, debe entenderse que las tareas se fueron ejecutando en paralelo con el desarrollo del curso, antes del cual no contaba con conocimientos en el área de Reconocimiento de Patrones.

Otras veces, las practicas incorrectas o poco recomendables corresponden a procedimientos establecidos de la empresa. Teniendo en cuenta que mi tesis de maestría abordará el mismo problema que estudiaremos en este informe, fue mi intención durante el desarrollo de este trabajo el interiorizarme en el proceso completo de armado de modelos tal como se hace hoy en día en la empresa para ganar conocimiento y proponer oportunidades de mejora sobre la base de lo ya existente.

Capítulo 1

Contextualización

PayGroup es una empresa uruguaya que ofrece soluciones de negocio para la industria financiera. Actualmente, uno de sus productos llamado "*RiskCenter*" es usado por varias entidades para el monitoreo de fraude transaccional. *RiskCenter* recibe una copia de las interacciones entre el cliente y la institución y genera una alerta cuando una de ellas resulta sospechosa. Luego un equipo de analistas de riesgo estudia las mismas y eventualmente toma medidas en función del riesgo de la alerta. Éstas podrían ser por ejemplo llamar al Tarjeta Habiente, bloquearle la tarjeta o enviarle un SMS advirtiendo de la transacción sospechosa. Tanto el proceso de análisis manual de las alertas como las medidas que se pueden tomar tienen un costo directo (sueldo de los analistas, costo de las llamadas, etc.) y un costo indirecto pero también relevante (molestias a los clientes, pérdida de confianza en la institución).

El negocio de pagos con tarjetas genera ganancia a través de un volumen muy grande de transacciones con poco margen de ganancia por transacción. Si bien los fraudes son fenómenos infrecuentes, éstos pueden reducir significativamente el margen de ganancia de la empresa por lo que hay un estímulo financiero para tratar de frenarlos. Sin embargo, el costo de análisis y medidas asociados a una alerta generan una tensión entre los extremos de "no se alerta ninguna transacción" (entonces no hay detección de ningún fraude) y "se alertan todas" (entonces se encuentra todo el fraude, pero a un costo de análisis insostenible).

Para poder determinar cuales son las transacciones que deben ser alertadas, *RiskCenter* utiliza un modelo de aprendizaje automático (comunmente denominado *modelo neural* en el ámbito financiero) que es distinto para cada institución financiera. Este proyecto versa sobre los pasos en la construcción de uno de estos modelos para un cliente específico de PayGroup.

La base de datos

Los datos para entrenar el modelo se encuentran almacenados en una base de datos de un total de 41.344.389 transacciones de tarjetas de crédito realizadas por clientes de un mismo banco, el cual emite tarjetas en tres países distintos. Todas fueron realizadas entre el 8 de Julio de 2014 y el 30 de Junio de 2015.

De cada transacción se cuenta con 81 campos, incluyendo a la etiqueta de clase (si la transacción es legítima o fraudulenta). Sin embargo, muchos de estos campos están vacíos o incompletos o no aportan información relevante por lo que la etapa de limpieza de datos reduce dramáticamente el número de características.

Desafíos del problema

Hay algunas particularidades de la base, de los datos o de las herramientas utilizadas para procesar la información que agregan dificultades al problema que normalmente no se presentarían o que

hacen que se requiera adoptar estrategias específicas. Detallamos las más importantes a continuación:

Dilución del fraude

El problema de detección de fraude es típicamente abordado como un problema de clasificación en dos clases: transacciones legítimas y fraudulentas. Sin embargo, las dos clases están muy desbalanceadas y esto debilita el desempeño de los clasificadores más frecuentes. Frente a este problema existen varias alternativas como ser submuestrear las transacciones de la clase mayoritaria, sobremuestrear las de la minoritaria o una combinación de ambas. En este informe veremos el resultado de aplicar algunas de estas técnicas a nuestros modelos.

A continuación presentaremos las proporciones de fraude en cada corte de la base. Por corte nos referimos a divisiones de los datos en grupos con realidades distintas.

Corte	Trxs	Fraudes	%Fraude	Dilución	% Fr.	% Fr. en Monto
País 9 EM 1	2.980.147	3.742	0,13 %	796	30, 2 %	20, 9 %
País 9 EM 2	32.509.735	3.605	0,01 %	9.018	29, 0 %	54, 6 %
País 10 EM 1	1.113.619	2.320	0,21 %	480	18, 7 %	6, 6 %
País 10 EM 2	4.072.430	2.395	0,06 %	1.700	19, 3 %	15, 8 %
País 11 EM 1	179.129	198	0,11 %	905	1, 6 %	0, 6 %
País 11 EM 2	489.329	151	0,03 %	3.241	1, 2 %	1, 6 %
TOTAL	**41.344.389**	**12.411**	**0,03 %**	**3.331**	**100 %**	**100 %**

Tabla 1.1: Distribución de los fraudes de la base en cada corte. En rojo: el corte prioritario. Por "Dilución" entendemos la proporción de legítimas versus fraudulentas. Con "Porcentaje de Fraudes en Monto" nos referimos a el porcentaje en dinero de las pérdidas totales por fraude en cada corte.

Como puede observarse, existe uno de los cortes en los que la proporción de fraude es sensiblemente peor que en el resto. Más aún: es el que tiene mayor volumen de transacciones legítimas y también en el que los fraudes cometidos son por montos mayores (más del 50 % de las pérdidas económicas se dan en ese corte), por lo que si bien es en el que se espera tener los peores resultados, también es el que requiere el mejor desempeño posible.

Tiempos de procesamiento de los datos

La tabla anterior también sirve para mostrar la gran cantidad de datos que se usan para entrenar los modelos. Esto es de por sí agrega trabas al problema pues implica que los archivos con el valor de cada variable para cada patrón sean de gran tamaño y demoren un tiempo muy grande en procesarse, aún para una cantidad de variables chica. Como en etapas de prueba se está constantemente cambiando las variables del problema, este proceso se vuelve considerablemente lento. Incluso es posible que para cantidad mayores de características, el archivo de datos sea inmanejable para el software que genera los modelos.

Capacidad del DE (Detection Engine)

Es también necesario tener en cuenta que todos los cálculos que deba hacer el modelo para determinar si se genera o no una alarma se tendrán que hacer *on-line* una vez que el modelo esté en producción. Como el tiempo de procesamiento es limitado (tipicamente, no es admisible demorar mas de un segundo por transacción) los atributos del modelo así como los algoritmos utilizados no deben ser excesivamente complicados para no dilatar los tiempos de procesamiento.

Además, los cálculos que realiza el DE no son tan complejos ni versátiles como los que uno hace, por ejemplo, en un motor de base de datos (*Oracle*, SQL, etc). Por lo tanto, cada vez que se construye un atributo debe tenerse en cuenta si el DE tiene la capacidad de calcular la nueva variable. De lo contrario, el modelo se vuelve imposible de colocar en producción.

Capítulo 2

Etapa Inicial

Este capítulo corresponde al armado del primer modelo neural propuesto. Si bien el desempeño del mismo resultó no ser satisfactorio, este capítulo es un ejemplo de un *pipeline* básico para armar un modelo de Aprendizaje Automático pues se cubren todas las etapas primordiales.

2.1. Limpieza de Datos

Como fue dicho anteriormente, el material bruto para la construcción del modelo neural es una base de datos con 81 atributos para cada transacción. Como parte del preprocesamiento de los datos fue necesario realizar un análisis de los campos de la base para determinar si los mismos son o no de utilidad. Para esto se estudió:

- **Presencia de datos faltantes o corruptos**: En esta primera etapa se quitaron atributos donde la cantidad de datos incorrectos fuera significativa (mayor al 30 %). Este filtro descarta una cantidad apreciable de campos, algunos muy útiles para problemas de este tipo. Dos ejemplos donde la calidad de dato está muy disminuida (34.676.022 de valores faltantes) son el número de cuenta y número de cliente correspondiente a la tarjeta. Lamentablemente, esto implica que en la gran mayoría de los casos es imposible determinar si dos tarjetas corresponden a la misma persona. A pesar de no estar completamente vacíos, se optó por no utilizar estos campos para no hacer asociaciones incompletas.

- **Sentido de los campos:** se investigó cual es el significado de cada atributo y se descartaron aquellos donde el mismo no es claro.

- **Coherencia entre sintaxis y semántica:** vinculado al punto anterior, se estudió que los datos representaran correctamente lo que se supone que deberían. Por ejemplo: para los campos que tienen codificaciones estandarizadas (Rubro, Moneda, País de la transacción etc), se investigó si los valores son coherentes con la misma. Esto implicó en algunos casos realizar consultas al banco para conocer el significado de algunas entradas. Para los atributos que deberían tener valores únicos, se estudió que no hubiera valores repetidos. También se descartaron atributos si tenían una distribución poco útil (por ejemplo, un valor que corresponde a más del 99 % de los casos y que engloba a todos los fraudes).

  En particular, para el campo "*monto de la transacción*" se buscó que no tuviera valores negativos, que fueran cantidades razonables y coherentes con la moneda de la transacción. A partir de este análisis se detectó un problema del banco en la traducción de este dato por lo que se optó por crear un campo estandarizado con los montos en dolares y corregido en los casos mal ingresados.

- **Asociación entre campos:** si existen campos que aplican restricciones unos a otros, se analizó si las cumplían. Por ejemplo: a cada tarjeta debe corresponderle un único país emisor.

Finalmente, en esta primera etapa fueron seleccionados los siguientes campos:

- Identificador de la transacción.
- Monto (en dolares).
- Fecha y hora.
- Tarjeta (dato encriptado).
- País emisor de la tarjeta.
- Código de comercio.
- País donde se efectuó la transacción.
- Modo de entrada (manual, chip o banda): este campo refiere a
- Código MCC (Merchant Category Code): es un código internacional estandarizado que determina el rubro de la transacción.
- IS_UPSCALE: 1 o 0 dependiendo de si la tarjeta pertenece o no a un grupo de clientes cuyo límite es mayor al normal.
- Tipo de transacción (débito o crédito).

2.2. Definición de cortes

Por cortes entendemos a divisiones del conjunto total de datos en grupos de modo de entrenar un modelo distinto para cada grupo. Esta es una práctica usual en PayGroup pues los clientes mismos solicitan que se entrene un modelo distinto para cada país donde tiene sucursales, en el entendido de que los consumidores tienen comportamientos distintos en cada país. Además de usar la división por país emisor (habiendo 3 paises distintos) se usó como variable de corte el modo de entrada, agrupándolo en dos conceptos: tarjeta presente o tarjeta no presente al momento de la compra. Hemos visto en la tabla 1.1 como se distribuyen las transacciones en cada corte.

2.3. Conjuntos de Train, CV y Test

Lo usual en el armado de modelos neurales en PayGroup es dividir el conjunto de transacciones de cada corte en los tres conjuntos nombrados de manera que la proporción de fraude en cada uno sea del 60 %, 20 % y 20 % respectivamente. Si bien existen muchas maneras posibles de hacer esta división, se optó por dividir el año en segmentos, de modo que por ejemplo, para el corte mayoritario el conjunto de entrenamiento corresponde a los meses entre Setiembre y Enero, CV (*Cross-validation*) corresponde a Febrero, Marzo y Abril y Test a Mayo y Junio. Es notorio que esta elección no parece ser la más adecuada, sobre todo si asumimos que el comportamiento de fraude no es estacionario en el año (lo cual parece razonable y puede comprobarse empíricamente), pero como hemos aclarado anteriormente se procedió de esta manera para interiorizarnos en los procesos establecidos de la empresa.

Es conveniente aclarar que las transacciones hechas en los meses de Julio y Agosto no se incluyen porque como veremos más adelante, algunas variables que utilizará el modelo encierran información de hasta dos meses hacia atrás. Esto implica que el cálculo de éstas en transacciones de esos meses no sería confiable porque no se tienen los datos históricos correspondientes.

Además, dado que las clases están extremadamente desbalanceadas en la base de datos, se aumentó artificialmente la proporción de fraudes realizando un submuestreo de transacciones legítimas de modo que el fraude represente el 10 % de las transacciones del conjunto de Train.

2.4. Generación de variables

Para el entrenamiento de un modelo neural, las variables simples que extrajimos son demasiado escasas y no contienen información sobre patrones de conducta, que es lo que nos interesa determinar para cada cliente. A continuación explicamos como usamos la base de datos para generar características que reflejen los comportamientos de cada cliente:

2.4.1. Variables manuales

Tres nuevas variables fueron creadas mediante la agrupación manual de valores de otras:

Grupo MCC

Incorporar la variable `MCC` tal como viene de la base de datos resultó ser problemático pues hay muchas categorías posibles (más de 500) y muchas de ellas con menos de un 0.1 % de las instancias. Por lo tanto, se decidió agrupar algunos de estos valores para llegar a una variable nominal con un grupo reducido de valores. Como criterio se utilizó en primer lugar, la semántica. Esto es: se agruparon códigos `MCC` que correspondieran a rubros similares. En segundo lugar, se estudió la distribución del fraude por rubro para agruparlos de manera que luego de la agrupación sea lo menos uniforme posible. Finalmente, se llegaron a las siguientes categorías:

Código	Descripción	%Trxs	%Fraude
A	Automoviles	14,2 %	34,4 %
C	Clothing	0,9 %	2,9 %
D	Variedad (department stores, etc)	14,5 %	8,8 %
E	Electronica / computacion	0,1 %	0,6 %
F	Cotidianos: Alimentacion/Farmacia	51,3 %	36,3 %
H	Articulos construccion	2,7 %	2,6 %
L	Luxury (joyería/relojes/discos)	0,8 %	1,2 %
P	Pagos	0,0 %	0,0 %
R	Retiros	6,1 %	3,9 %
S	Suscripciones	0,0 %	0,0 %
T	Telefonía Celular	1,0 %	1,2 %
V	Agencias de viaje	0,1 %	0,1 %
W	Hobbies / Entretenimiento / Tiempo Libre	0,4 %	2,3 %
X	Aerolíneas	0,0 %	0,0 %
0	Sin clasificar	7,7 %	5,8 %

Tabla 2.1: Distribución del fraude en las categorías de la variable `MCC_Group`. Corte: País 9 EM 2.

Grupo de monto

Esta variable se armó mediante la agrupación de valores del monto. El objetivo fue aislar comportamientos observados en algunas de estas franjas. Por ejemplo: el comportamiento del fraude en montos pequeños es muy distinto que en montos mayores, ya que es conocido que en compras en internet, un conjunto de muchas transacciones seguidas por un monto muy bajo correspondan a un fraudador "probando" la tarjeta para verificar si la misma está operativa. Las categorías finales son la siguientes:

Código	Descripción	%Trxs	%Fraude
MI	De 0 a 5 dolares	18,6 %	6,5 %
BA	De 5 a 10 dolares	17,5 %	4,5 %
BM	De 10 a 20 dolares	19,5 %	8,2 %
ME	De 20 a 50 dolares	25,0 %	21,1 %
MA	De 50 a 100 dolares	10,8 %	17,8 %
AL	De 100 a 500 dolares	8,0 %	31,0 %
AX	De 500 a 2000 dolares	0,6 %	9,1 %
MX	Más de 2000 dolares	0,1 %	1,9 %

Tabla 2.2: Categorías de la variable `Amount_Group`. Corte: País 9 EM 2.

Franja horaria

Observando casos particulares, fue posible determinar que existían momentos del día (como la madrugada) donde la cantidad de compras bajaba notoriamente. Por este motivo se decidió agrupar las horas del día en categorías para crear una variable nominal que ayudara a describir el patrón de compras de un cliente según el momento del día (como veremos más adelante: para usarse como un concepto de acumulación). El resultado es este:

Franja Horaria	%Trxs	%Fraude
De 01:00 a 05:59	4 %	6 %
De 06:00 a 08:59	5 %	4 %
De 09:00 a 10:59	8 %	9 %
De 11:00 a 14:59	27 %	30 %
De 15:00 a 19:59	42 %	37 %
De 20:00 a 21:59	8 %	8 %
De 22:00 a 00:59	6 %	6 %

Tabla 2.3: Distribución del fraude en las categorías de la variable `Time_Range`. Corte: País 9 EM 2.

2.4.2. Acumuladores

Para generar alertas, *RiskCenter* tiene la capacidad de calcular "acumuladores" en el momento de recibir cada transacción. Los acumuladores permiten considerar comportamiento previo al momento de evaluar una transacción generando variables adicionales. La gran flexibilidad de la herramienta hace que se cuente con un vasto pool de variables pues a priori hay posibilidades casi infinitas de elección de conceptos de acumulación. Presentaremos ahora los que utilizamos en este problema en particular. En definitiva, cada variable se generó mediante la combinación de uno o varios elementos de las siguientes categorías:

Conceptos de acumulación (fijos o variables):

- Modo de entrada
- Grupo MCC
- Grupo de Monto
- Franja Horaria
- Comercio
- País

Período de tiempo:

1 hora, 24 horas, mismo día, 7 días, 31 días o 62 días.

Funciones de agregación:

Cantidad, Suma, Promedio, Máximo, Mínimo o Desviación Estándar.

Cortes:

Generados a partir de los mismos campos que los conceptos de acumulación.

Luego de seleccionados los componentes, se construye una consulta de `SQL` a la base y se arma un script que calcula el atributo. Algunos ejemplos de variables posibles son:

- *Suma de los montos de transacciones de la tarjeta en el mes anterior.*

  Generada sin un concepto o corte específico, período = 31 días, función de agregación = suma.

- *Cantidad de transacciones en el mismo comercio en la última semana.*

  Generada con: concepto = comercio (variable) , período = 31 días, función de agregación = suma, sin función de corte.

- *Desviación estándar de los montos de compras en supermercados hechas en el día de hoy, excluyendo las compras electrónicas.*

  Generada con: concepto = grupo MCC (fijo), período = mismo día, función de agregación = desviación estándar, corte = modo de entrada.

Como se ve, con un concepto de acumulador fijo nos referimos a acumular, por ejemplo, para un país determinado. Acumulador variable podría ser uno que según el contexto, elija determinados países (el país de cada transacción, por ejemplo).

Además de las variables que se pueden armar con los criterios anteriores, resulta interesante considerar cocientes de las mismas para poder capturar información comparativa y estudiar patrones de comportamiento. Sin embargo, en lugar de considerar cocientes simples de la forma x/y (donde x e y fueron calculadas mediante acumuladores) se utilizó la siguiente función:

$$f(x,y) = \begin{cases} 2 - \dfrac{y}{x} & \text{si } x > y > 0 \\ \dfrac{x}{y} & \text{si } y > x > 0 \\ -\dfrac{1}{y} & \text{si } x = 0, y > 1 \\ y - 2 & \text{si } x = 0, y \leq 1 \end{cases}$$

f tiene como pre-condiciones que $y \geq 0$, $x \geq 0$ y que $y = 0$ implica $x = 0$, lo cual tiene sentido para un gran número de comparaciones interesantes. Por ejemplo: supongamos que queremos comparar un concepto de acumulación (e.g.: compras en telefonía celular) en la última hora versus en los últimos dos meses. Esto podría servir para detectar si un cliente se está desviando notoriamente y de manera muy rápida de su patrón de compra habitual. Tenemos entonces que si llamamos x a la acumulación en una hora e y a la acumulación en 62 días, se cumple siempre que:

- Para las funciones de agregación Suma y Máximo: y es siempre mayor o igual a x.

- Para la función de agregación Promedio y Desviación Estándar: puede pasar que y sea menor a x, pero $y = 0$ implica $x = 0$.

La función mínimo queda excluida pues no cumple con la condición. Estas aclaraciones aplican siempre que x e y utilicen la misma función de agregación. Sin embargo, es posible extender el razonamiento a otros casos (por ejemplo: comparar mínimo x versus máximo y por el mismo concepto en el mismo período de tiempo).

Las ventajas de utilizar esta función son:

- Al usar el cociente simple x/y se pierde la información de y cuando $x = 0$, lo que no pasa usando f.
- El codominio de f es [-2,2] por lo que todas las variables obtenidas de este modo existen en espacios con métricas comparables.
- Al contemplar los casos de divisiones entre 0 se evitan errores de cálculo.
- La visualización de los datos es más completa y pueden reconocerse rápidamente varios casos.

2.5. Selección de características

Luego de generar un pool de características es momento de seleccionar para cada corte cuales son las relevantes. Describiremos a continuación los pasos del proceso centrándonos en lo acontecido en el corte mayoritario:

Ranking de atributos por Information Gain (IG)

En primer lugar se buscó utilizó un criterio que permitiera reducir rápidamente el inmenso universo de características descartando aquellas que eran evidentemente no significativas. Teniendo en cuenta que siempre se estaba esperando entrenar mediante un Random Forest, fue natural realizar un ranking de variables según el criterio de Information Gain pues es el mismo que utilizan las implementaciones clásicas RF para elegir atributos en cada paso. Además, el IG tiene ventajas claras con respecto a el Gain Ratio en un caso de clases con fuerte desbalance como es el nuestro, pues el último tiende a verse favorecido para atributos con entropías cercanas a 0, algo que es habitual y que puede significar escaso poder de discriminación.

El ranking se armó en el conjunto de entrenamiento utilizando el algoritmo `InfoGain` de `WEKA` con el criterio de rankeado `GeneticSearch` con 40 generaciones. Del conjunto de aproximadamente 1000 variables iniciales se descartaron 200 por tener valores de IG menores a 0.001.

Selección manual por criterio semántico

Luego del paso anterior, aún resta un pool de aproximadamente 800 variables. Sin embargo, muchas de ellas resultan muy redundantes, ya que se obtienen por pequeñas variaciones en el cálculo. En esta etapa, el criterio utilizado fue agrupar las variables por un criterio semántico (que se hayan obtenido mediante acumuladores del mismo concepto) y quedarse dentro de cada grupo con las 2 o 3 mejores rankeadas en el listado anterior. Por ejemplo: dentro del concepto de acumulación "compras" (considerar todos los rubros menos Pagos y Retiros), las mejores rankeadas fueron la suma, el promedio y el máximo en las últimas 24 horas, por lo que nos quedamos con estas 3. Gracias a este criterio, mitigamos la redundancia en las variables y redujimos a aproximadamente 100 variables por corte.

Wrapping manual

A esta altura, para el modelo prioritario los resultados en CV habían mejorado pero no eran satisfactorios. Finalmente, se logró mejorar el desempeño reduciendo la cantidad anterior de atributos hasta llegar a un total de 49. El método utilizado fue un *Wrapping*: se evaluó sucesivamente el

modelo resultado de remover las características en pequeños grupos de acuerdo con los dos criterios anteriores, esto es, sacando primero las de menor IG y mayor redundancia semántica.

Es conveniente notar que estos criterios no pretenden ser óptimos sino que priorizan la sencillez y velocidad para reducir el universo de características a un tamaño manejable. Como hemos aclarado anteriormente, este proceso se llevó a cabo en una etapa muy inicial por lo que existen muchas alternativas posibles para mejorarlo.

2.6. Primer Entrenamiento

Luego de obtener un universo de características reducido con un desempeño mejorado en CV, se procedió a evaluar los resultados en el conjunto de test. Para esto, se entrenó usando `WEKA` un total de 10 Random Forests con semillas distintas (para mitigar efectos aleatorios del algoritmo en el desempeño) donde cada bosque estaba formado por 50 árboles con profundidad liberada y eligiendo en cada etapa la mejor variable de entre 22 atributos sorteados al azar de las 49. La medida de desempeño utilizada es una variación de la curva PRC, donde cambiamos la medida de Precisión por su inversa, obteniendo así la cantidad de Falsos Positivos (FP) generados por cada fraude alertado. Esta medida junto con la Efectividad son las dos que nos interesan en el contexto del modelo para determinar si es bueno o no.

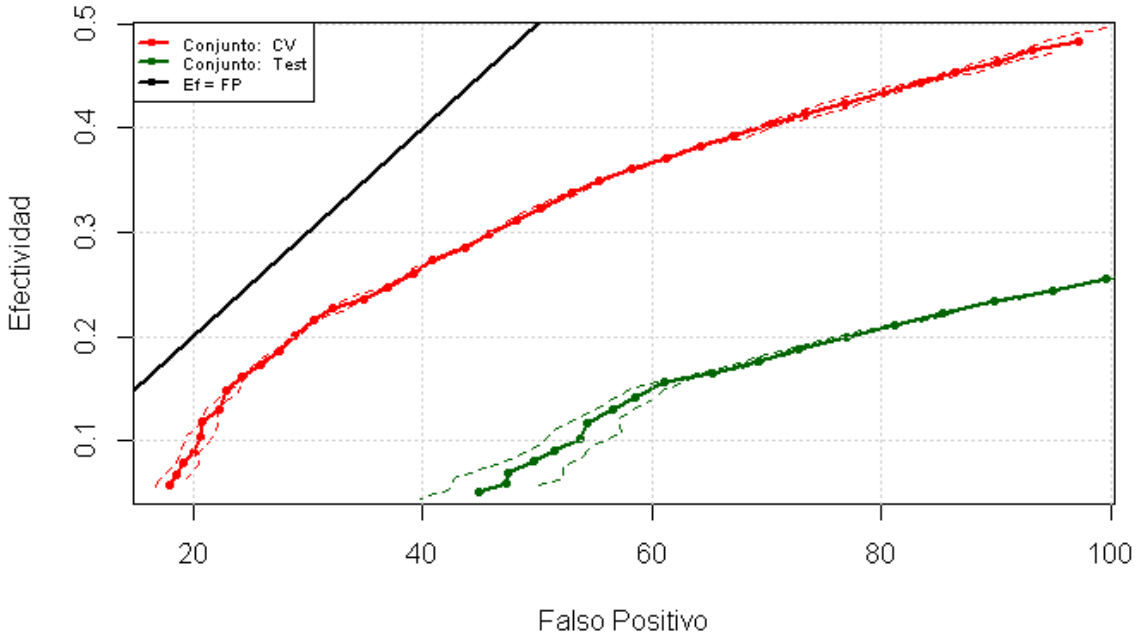

Figura 2.1: Efectividad versus FP para el modelo del corte mayoritario en la primera etapa. Las líneas gruesas son los promedios, mientras que las punteadas corresponden a las curvas más y menos optimistas (extremos de un intervalo de confianza calculado al 99 % para la media).

Como puede observarse, no solamente los resultados en Test son muy inferiores a los de CV, sino que son absolutamente inaceptables para el contexto del problema: pensemos que para este cliente las expectativas rondan valores de efectividad del 60 % con un FP de 20:1. Por lo tanto se hizo necesario seguir estudiando el problema para mejorar los modelos.

Capítulo 3

Segunda Etapa

Teniendo en cuenta los resultados primarios obtenidos se decidió proceder con un diagnóstico del clasificador. Una manera clásica de estudiar si se está incurriendo en Underfitting o Overfitting es estudiar la evolución del error a medida que aumentan las muestras con las que se entrena el clasificador. El resultado de esta prueba podemos verlo a continuación:

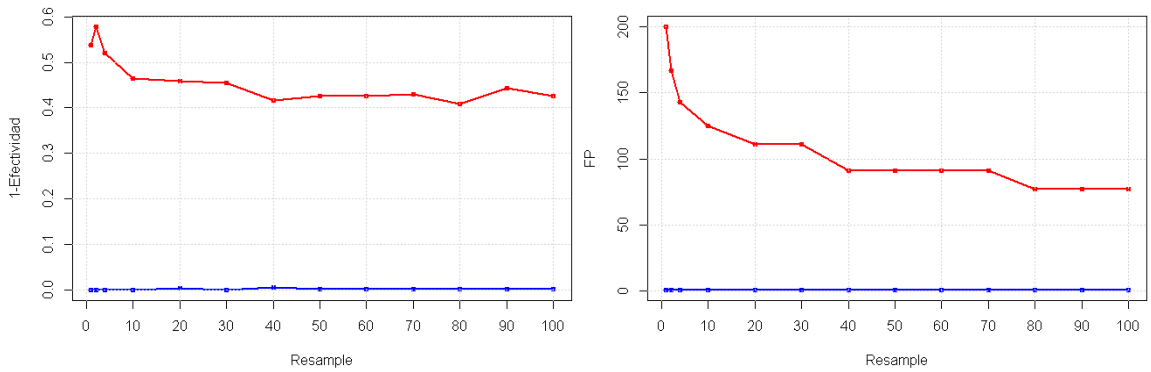

Figura 3.1: Porcentaje de instancias de Train usadas para entrenar versus dos medidas de error distintas. En azul: el error en Train. En rojo: el error en CV.

Como puede verse estamos en una situación típica del sobreajuste pues el error en Train es pracitcamente nulo mietnras que el error en CV tiene una tendencia general descendente. Esto tiene sentido pues si el clasificador es demasiado específico tendrá problemas para generalizar pero cuanto más instancias de entrenamiento le demos, más general se volverá a pesar del sobreajuste.

En las siguientes secciones mostraremos los pasos que se siguieron para intentar de solucionar este problema y afinar el modelo.

3.1. Selección de parámetros de RF

El clasificador Random Forest, al igual que casi cualquier otro algoritmo, tiene sus propios parámetros que pueden afectar drásticamente el desempeño de los clasificadores construidos. En particular, los tres parámetros sobre los que `WEKA` permite un control son:

- `depth`: profundidad máxima de los árboles del bosque.
- `NumArboles`: número de árboles que componen el bosque.

- `NumAttrib`: número de atributos que eligen aleatoriamente en cada nodo. De estos, se tomará el mejor rankeado por el criterio de construcción del árbol (IG, Gain Ratio, etc) para armar los próximos nodos.

Una revisión de la etapa inicial determinó que los mismos no fueron determinados por ningún criterio específico más que el ensayo y error en unos pocos casos. Esto es: mientras se realizaba la selección de características, se dieron valores iniciales que fueron cambiados durante el transcurso de la evaluación por lo que los valores finales fueron pensados para un conjunto de características obsoleto. En consecuencia, se procedió a optimizar los mismos.

La metodología consistió nuevamente de un wrapping, evaluando el desempeño de 10 repeticiones con semillas distintas del clasificador para cada punto del dominio del parámetro. Sin embargo, esto genera una enorme cantidad de pruebas: tan solo encontrar el valor óptimo de `depth` implica entrenar 10 clasificadores para cada valor entre, por ejemplo, el intervalo 10 a 50. Como obtener y evaluar la curva PRC completa para cada instancia hubiera sido demasiado costoso computacionalmente, se adoptó la estrategia de evaluar los valores de Efectividad y Falso Positivo solamente para un valor de umbral de 0,5. [1] A continuación presentamos los resultados de graficar los parámetros vs las variables FP y 1-Efectividad (de esta manera comparamos medidas de *error*, y por lo tanto buscamos mínimos de las curvas):

Profundidad

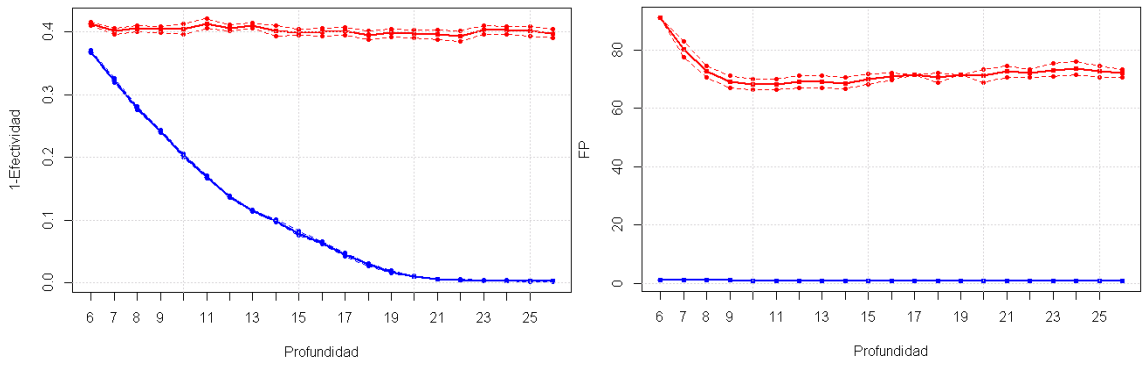

Figura 3.2: Gráfica de dos medidas de error distintas versus el parámetro `depth`. En azul: el error en Train. En rojo: el error en CV. Las líneas centrales son los promedios, mientras que las punteadas corresponden a los extremos de un intervalo de confianza calculado al 99 % para la media. No se consideran profundidades menores a 6 por cuestiones de escala.

La profundidad de los árboles afecta directamente la complejidad del modelo, por lo que el comportamiento del error es el esperable de acuerdo con la teoría general: para valores muy bajos, el modelo no es lo suficientemente rico por lo que el error es alto y se cometen equivocaciones similares en Train y CV. En cambio para valores altos los arboles se hacen demasiado extensos y se entrenan nodos con una cantidad de instancias que no es estadísticamente significativa, por lo que el modelo resultante se ajusta muy bien al conjunto de Train pero no logra generalizar bien al conjunto de CV.

Teniendo en cuenta que la profundidad estaba liberada en nuestro modelo anterior, nos encontrábamos en valores muy a la derecha en los gráficos por lo que obtenemos otro indicio de que estábamos incurriendo en un sobreajuste del modelo. Finalmente decidimos tomar un valor de `depth`

[1] Con el umbral nos referimos al sistema utilizado por `WEKA` en donde a cada instancia se le asigna un puntaje o *score* dependiendo de cuantos árboles del bosque lo clasificaron como fraude. Las instancias cuyo score supere cierto umbral son marcadas como fraude. Por defecto, `WEKA` devuelve un archivo en formato `.out` cada vez que realiza un entrenamiento, y en el mismo las medidas de Efectividad y Precisión para el umbral por defecto de 0,5. Podemos pensar que estamos evaluando entonces la curva Efectividad vs FP en un punto específico.

= 10 pues se corresponde con el mínimo absoluto de la curva roja en la derecha y corresponde a una efectividad en Train de 80 % que parece razonable si el modelo no está sobreajustado.

Una vez elegido este valor, se procedió a optimizar el siguiente parámetro:

Número de árboles del bosque

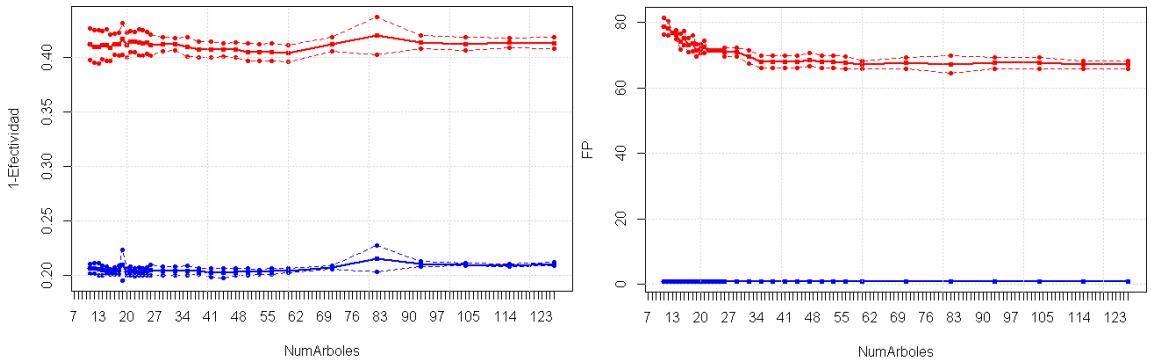

Figura 3.3: Gráfica de dos medidas de error distintas versus el parámetro `NumArboles`. En azul: el error en Train. En rojo: el error en CV. Las líneas centrales son los promedios, mientras que las punteadas corresponden a los extremos de un intervalo de confianza calculado al 99 % para la media.

En este caso no es tan claro que el comportamiento para valores muy altos sea el de un modelo sobreajustado, aunque si es cierto que a partir de cierto número de árboles el desempeño no parece mejorar significativamente. Por lo tanto, optamos por un valor de `NumArboles` = 37 para no complejizar en exceso el modelo

Número de atributos por nodo

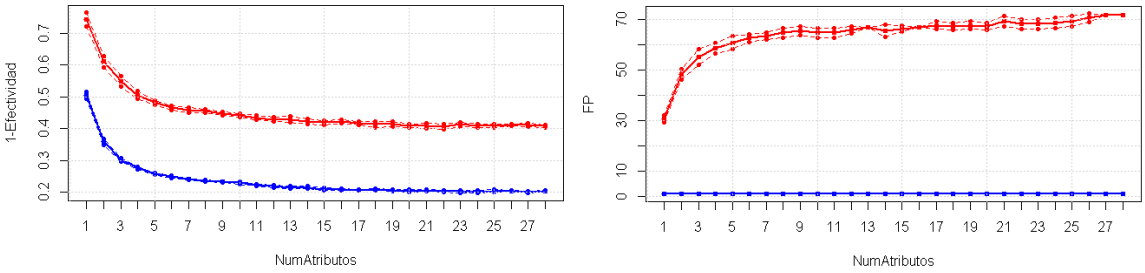

Figura 3.4: Gráfica de dos medidas de error distintas versus el parámetro `NumAttrib`. En azul: el error en Train. En rojo: el error en CV. Las líneas centrales son los promedios, mientras que las punteadas corresponden a los extremos de un intervalo de confianza calculado al 99 % para la media.

Como puede verse, existe un compromiso en la elección de este parámetro, pues disminuir un tipo de error aumenta el otro. La decisión finalmente fue tomar `NumAttrib` = 6 que es muy cercano al valor por defecto que da `WEKA`[2] y es el punto donde ambos errores comienzan a estabilizarse en CV.

[2]El valor por defecto es $\log_2$(cantidad de atributos).

3.2. Nueva selección de características

Dado que se sospechaba un sobreajuste, es posible que hubiera variables que son redundantes entre sí y agregan ruido o que resultan demasiado específicas y confunden al modelo más de lo que aportan. De esta idea surgió la posibilidad de realizar una selección de características más exhaustiva sobre el conjunto de variables del problema. En particular, se buscó determinar un subconjunto de los 49 atributos del modelo anterior cuyas variables estén poco correlacionadas entre sí y que entrenen un modelo con mejor desempeño. Como una búsqueda exhaustiva es impracticable aún para esta cantidad muy reducida, se optó por un *Backward Selection* con esta estructura:

1. Remover atributos de a uno por vez y entrenar.
2. Agrupar las variables que al quitarlas mejoran el desempeño en un conjunto A. Si no hay, pasar a 5.
3. Si la cantidad de subconjuntos de A es razonable, realizar una remoción exhaustiva. De lo contrario, quitar subconjuntos de A con alta correlación entre sí. Ejecutar la decisión óptima de las estudiadas.
4. Repetir 1.
5. Estudiar la correlación entre los atributos restantes. Estudiar si es posible remover variables sin afectar o mejorando el desempeño.
6. Si se ejecutaron cambios en el paso anterior, volver a 1. De lo contrario, finalizar.

Nuevamente en cada paso la evaluación de desempeño se realiza entrenando 10 modelos distintos por cada conjunto de atributos a remover y estudiando los valores de FP y 1 - Efectividad para el umbral 0,5. Luego de la ejecución del algoritmo anterior se obtuvo un conjunto final de 29 atributos.

3.3. Segundo Entrenamiento

Luego de aplicar estos cambios volvimos a hacer una evaluación en el conjunto de CV y en el Test. Los resultados son los siguientes:

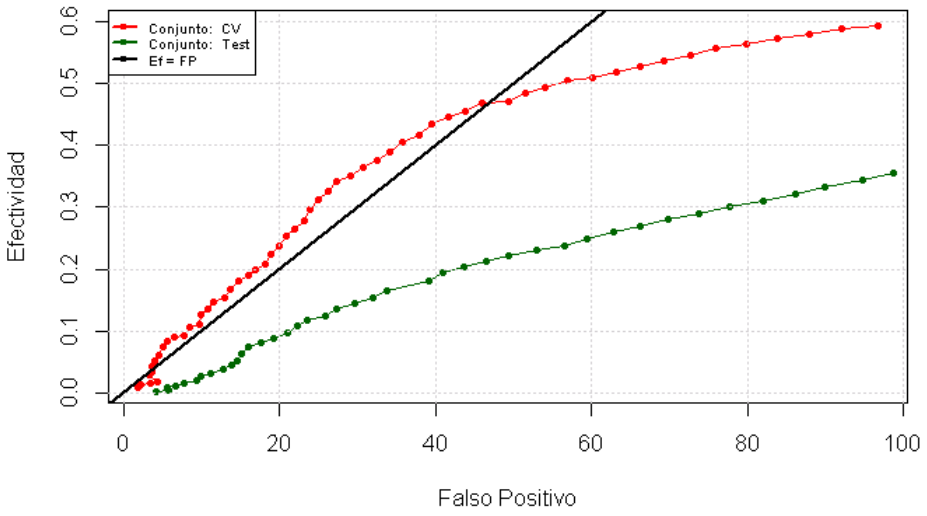

Figura 3.5: Efectividad versus FP para el modelo del corte mayoritario en la segunda etapa.

Puede verse que el desempeño a mejorado notablemente, pero aún está lejos de parecerse a algo que esté cerca de las expectativas del cliente. Por lo tanto, se hace necesario seguir trabajando.

Capítulo 4

Tercera Etapa

Frente a la posibilidad de que existiera un sobreajuste, el desempeño en el conjunto de CV se vuelve poco indicativo pues ya sabemos que está afectado por el exceso de especificidad. Es por eso que se optó por utilizar un nuevo conjunto de CV: si bien este debería haber incluido los meses de Febrero y Marzo, por cuestiones de tamaño en disco de los archivos y velocidad de procesamiento sólo se utilizó el primer mes durante la etapa inicial. Por lo tanto, el modelo no está ajustado para el mes de Marzo y puede usarse como un nuevo conjunto de CV.

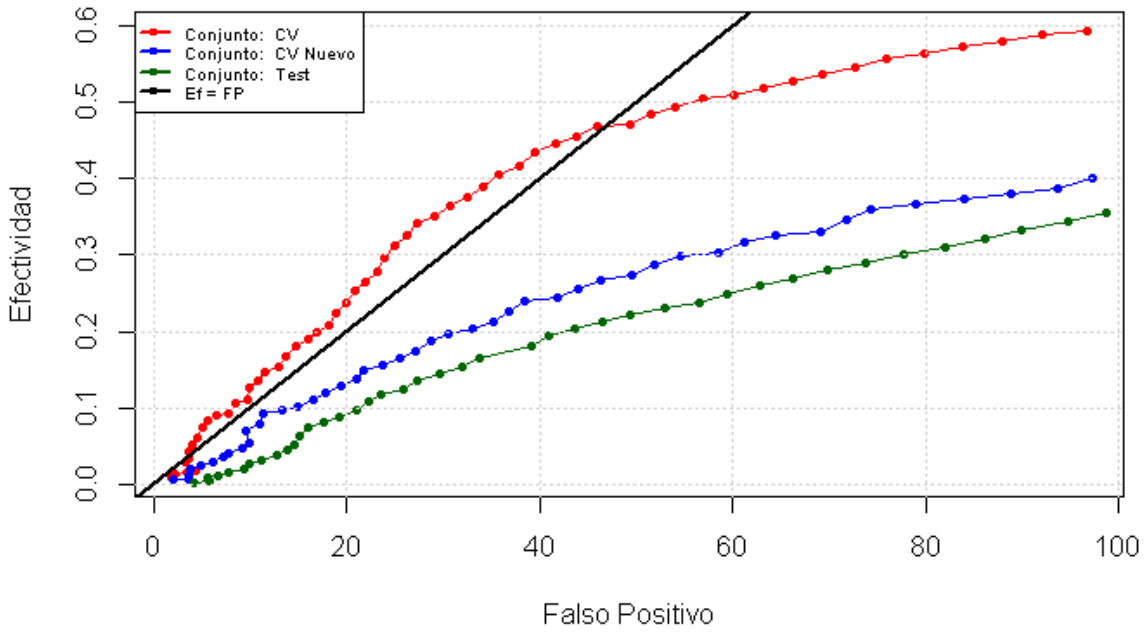

Figura 4.1: Efectividad versus FP para el modelo del corte mayoritario al final de la segunda etapa.

De hecho, al no haberse evaluado jamás en este conjunto hasta el final del armado del modelo, el desempeño esperado es similar al del conjunto de Test (como puede verse en la gráfica) por lo que cualquier cambio que aumente el desempeño en el nuevo CV sería esperable que también lo haga en Test.

En las siguientes secciones realizamos nuevos experimentos buscando corregir aún más el desempeño.

4.1. Inclusión de variables

Recordemos que en la etapa anterior se realizó un Backward Search para seleccionar el mejor conjunto de atributos. Este procedimiento es claramente suboptimo pues como adopta una estrategia ávida, al llegar a un óptimo local no revisa las decisiones tomadas para buscar alternativas. Como una manera de buscar acercarse más a un resultado óptimo, se realizó un Forward Search en el conjunto final de la etapa 1 (consistente de 49 atributos) partiendo del pool hallado en la parte 2. La estrategia fue similar: se incluyeron una por vez las variables, se estudió el desempeño mediante la evaluación en CV y se analizó el efecto de agregar cada subconjunto posible del conjunto de las variables que individualmente daban mejoras. En esta ocasión sin embargo esta estrategia no conllevó grandes mejoras pues tan solo unas 3 variables fueron nuevamente incluidas en el conjunto de atributos.

4.2. Algoritmos Potenciadores

En esta etapa decidimos investigar un conjunto de potenciadores de clasificadores que están implementados en `WEKA` y que son de fácil utilización. Los mismos los mostramos a continuación:

`AdaBoost:`

Este algoritmo es considerado como sumamente eficaz para mejorar el rendimiento de clasificadores débiles. La idea básica es entrenar múltiples modelos que a pesar de no ser tan buenos, son combinados linealmente con pesos que ayuden a maximizar el desempeño de la salida.[1] El parámetro del algoritmo que consideramos para el estudio del desempeño fue `Iterations` (el número de modelos débiles a combinar). El resultado al evaluar desempeño para umbral 0,5 es el siguiente:

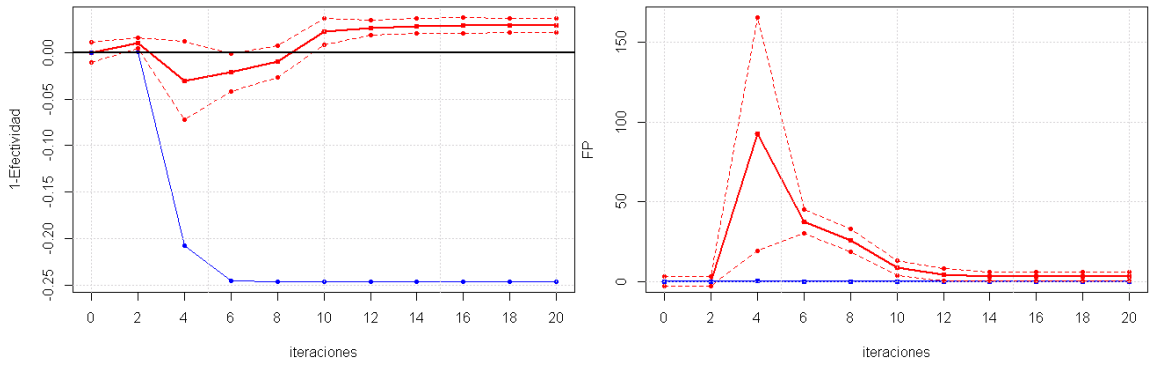

Figura 4.2: Gráfica de variaciones en las dos medidas de error versus el parámetro `Iterations` (0 corresponde a no aplicar AdaBoost). En azul: el error en Train. En rojo: el error en CV. Las líneas centrales son los promedios, mientras que las punteadas corresponden a los extremos de un intervalo de confianza calculado al 99 % para la media. La línea negra horizontal representa la variación cero.

Vemos que a pesar de la potencia del método, en nuestro modelo no da buenos resultados. Esto puede explicarse teniendo en cuenta que bajo ciertas hipótesis, se ha discutido que AdaBoost puede tender al overfitting (por ejemplo, ante la presencia de datos "confusos" [2]). Como nuestro modelo probablemente ya sufre de alta varianza tendría sentido que el problema empeore.

[1] Ver capítulo 3 de [1] para una explicación detallada.

`Bagging:`

Técnicamente, este algoritmo ya se utiliza dentro del Random Forest pues RF es un Bagging de Árboles de Decisión (ver capítulo 15 [1]). Sin embargo, esta redundancia no resulta perjudicial como veremos a continuación:

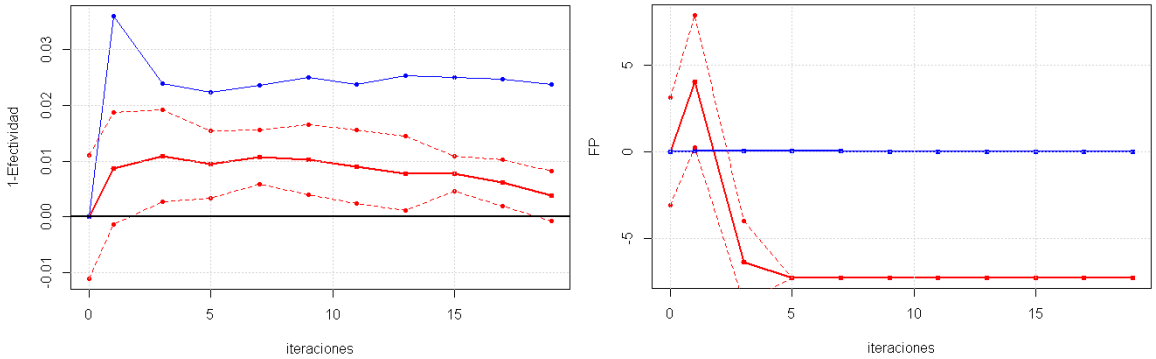

Figura 4.3: Gráfica de variaciones en las dos medidas de error versus el parámetro `Iterations`.En azul: el error en Train. En rojo: el error en CV. Las líneas centrales son los promedios, mientras que las punteadas corresponden a los extremos de un intervalo de confianza calculado al 99 % para la media. La línea negra horizontal representa la variación cero.

Nuevamente lo que graficamos son las variaciones con respecto a `Iterations`=0 (correspondiente a no aplicad el algoritmo). En este caso si vemos una mejora notoria en lo que respecta a Falsos Positivos, y a un costo de Efectividad muy bajo. Concluimos que es una buena idea aplicar Bagging a nuestro clasificador de ahora en más.

4.3. Manipulación del submuestreo

Luego de aplicar las mejoras de las etapas anteriores se decidió profundizar en el problema de la composición del conjunto de entrenamiento. El sobremuestreo de fraudes realizado por `SMOTE` suele ser recomendado para problemas de clases desbalanceadas, pero como vimos anteriormente el conjunto de Train se armó submuestreando transacciones legítimas para que el fraude esté en proporción de 0,1 contra las legítimas. En estas condiciones realizar un sobremuestreo del fraude parece no ser una buena idea, porque llegaría un conjunto demasiado alejado del original.

Para investigar sobre esta cuestión se realizó un experimento en el cual se entrenó el clasificador con conjuntos de entrenamiento a los cuales se les disminuyó el submuestreo de modo que la proporción de fraudes en cada conjunto tomara valores progresivamente más pequeños. El valor mínimo que se tomó fue de $0,03$ pues valores más pequeños implican un tamaño para el conjunto de Train que se vuelve inmanejable. Para cada conjunto de Train nuevo estudiamos el desempeño en CV al entrenar.

Como vemos en la figura 4.4, si bien las curvas no están completamente separadas, a medida que disminuimos la proporción de fruade y aumentamos la cantidad de legítimas en Train la tendencia general es a mejorar el desempeño, por lo que finalmente nos quedamos con un conjunto de Train submuestreado con una proporción de 0,03.

Una vez que llegamos a la cantidad de transacciones legítimas óptima aplicamos el algoritmo `SMOTE` y buscamos la cantidad de fraudes que debemos generar artificialmente para obtener una mejora máxima. Probamos con crear cantidades iguales al 50 %, 100 % y 200 % del total de fraudes respectivamente. Sin embargo, en este caso la conclusión es que aumentar la cantidad de fraudes no mejora significativamente al modelo. Por lo tanto, prescindimos de este algoritmo.

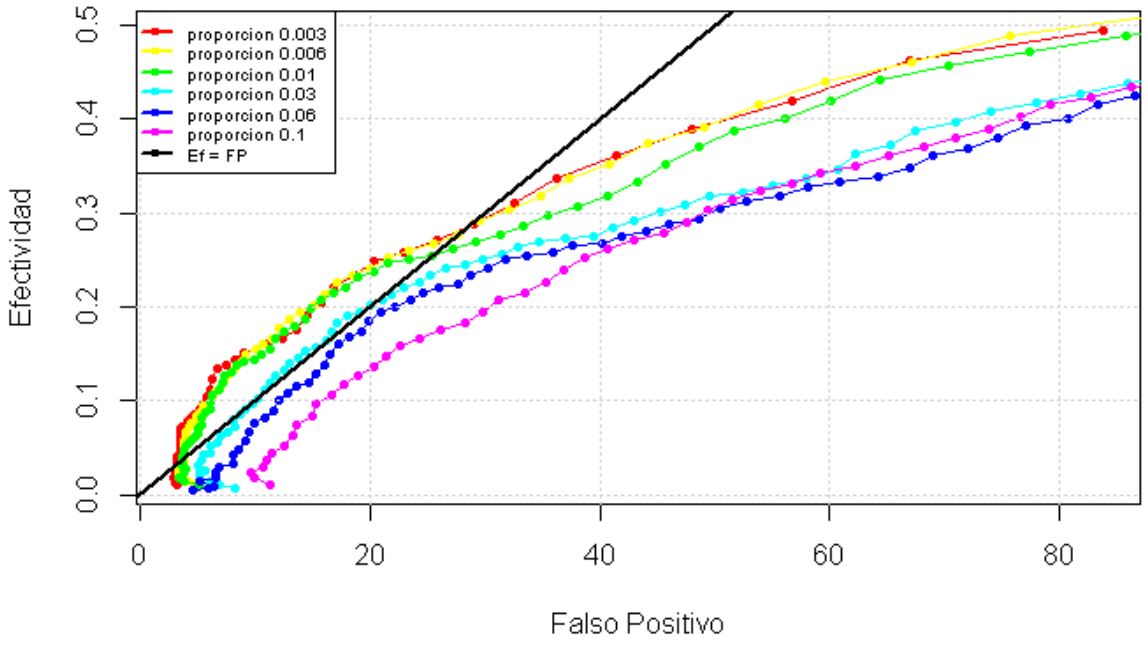

Figura 4.4: Efectividad versus FP para distintos submuestreos del conjunto de Train. Cada curva se obtuvo promediando los desempeños de 10 semillas distintas.

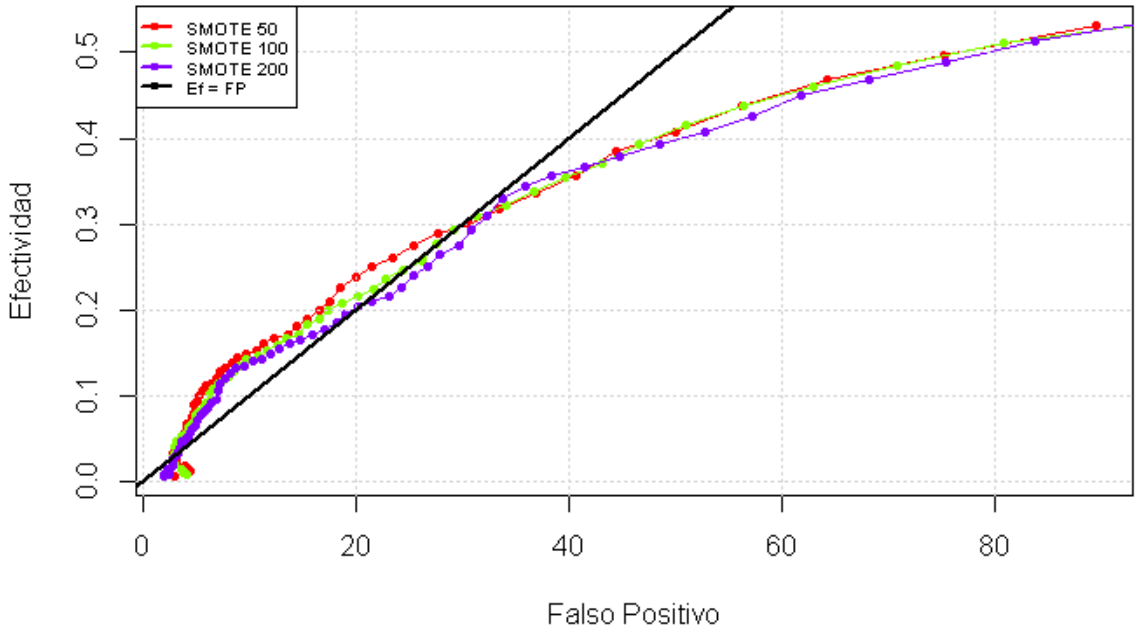

Figura 4.5: Efectividad versus FP para distinta cantidad de fraudes generados artificialmente. Cada curva se obtuvo promediando los desempeños de 10 semillas distintas.

4.4. Tercer Entrenamiento

Luego de aplicar de forma conjunta todas las estrategias que presentamos (cuyo efecto individual era leve), realizamos un tercer entrenamiento cuyo desempeño en el nuevo conjunto de CV es el que vimos anteriormente. De forma más clara puede visualizarse a continuación:

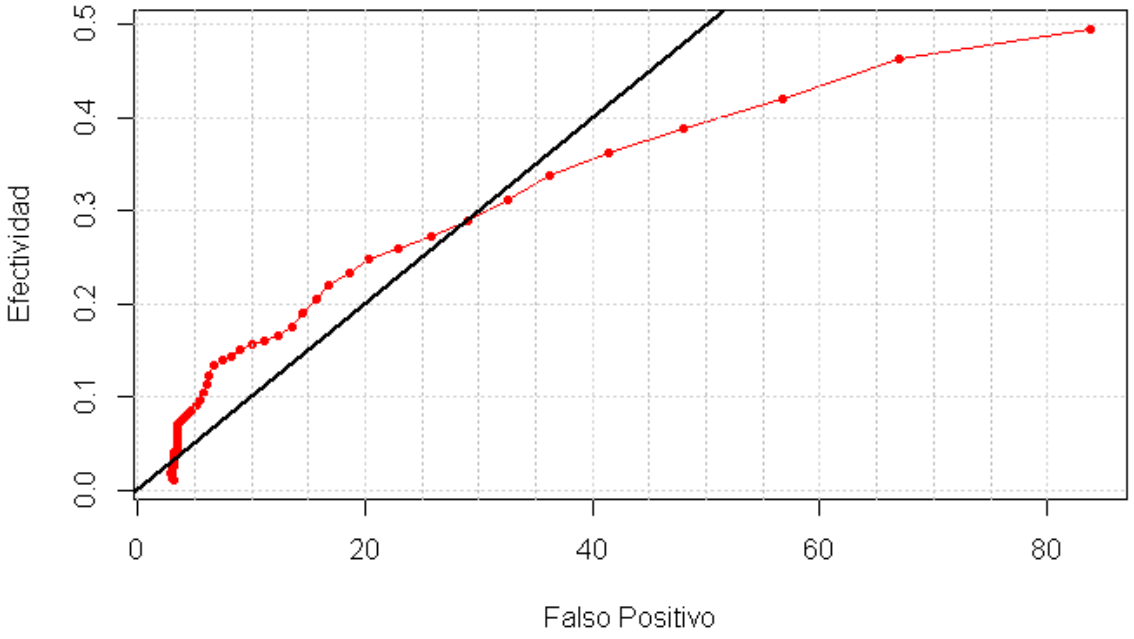

Figura 4.6: Efectividad versus FP para el modelo del corte mayoritario al final de la tercera etapa. Evaluación en el conjunto de CV correspondiente a Marzo.

Es de notar que este desempeño es muy parecido al que se tenía para el conjunto de CV de Febrero al final de la etapa 2, por lo que comparando con la gráfica 4.1 vemos una mejora significativa para el conjunto de CV de Marzo. Tal como dijimos anteriormente, es esperable que el desempeño en Test no difiera en gran medida del desempeño en este conjunto de CV. De todas maneras, el resultado sigue siendo inferior al esperado por lo que se continuarán realizando pruebas.

Capítulo 5

Pruebas Finales

Los experimentos que realizamos en esta sección son íntegramente motivados por el curso de Reconocimiento de Patrones. Aunque son los finales en el presente trabajo, de ninguna manera son los últimos que haremos.

En una nueva línea de pensamiento, buscamos en esta etapa aplicar algoritmos de clustering. Esta práctica está basada en un supuesto usual cuando se aborda el estudio del fraude en tarjetas de crédito que es que un fraude es un outlier dentro del comportamiento usual del cliente. De esta manera, un clustering podría darnos pistas de alguna estructura intrínseca en el conjunto de datos que no hayamos encontrado en etapas anteriores.

Detectar la presencia de grupos significativos podría servir para armar nuevos cortes o generar nuevas variables (por ejemplo: una variable categórica indicando el número de cluster). En los experimentos específicos que haremos ahora, buscaremos que el clustering nos de pistas acerca de la distribución de los fraudes con respecto a las legítimas en espera de detectar fraudes "ruidosos". Para entender con que nos referimos a esto observemos el siguiente ejemplo artificial:

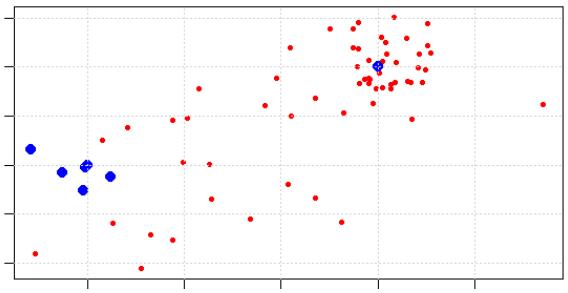

Figura 5.1: Ejemplo sintético. En rojo: transacciones legítimas. En azul: fraudes.

En este caso, es claro que un clasificador podría detectar fácilmente una frontera de decisión para el grupo de fraudes de la izquierda sin cometer mayores equivocaciones en las legítimas. Sin embargo, el costo de detectar el fraude que está aislado del resto es muy alto pues implica un alto número de falsas alarmas (si no se hace un sobreajuste). Si esta fuera la situación en nuestro conjunto de datos, podría ser preferible descartar una pequeña cantidad de estos fraudes aislados, sacrificando en ello la Efectividad pero bajando quizás el Falso Positivo en una medida mucho mayor.[1] Para saber si existen fraudes de este tipo tomamos dos abordajes distintos que presentamos a continuación.

[1] Teniendo en cuenta que el archivo de Train sobre el que estamos ejecutando este algoritmo ya tiene un submuestreo, podemos interpretar como que los siguientes experimentos son un segundo submuestreo en el que elegimos los datos más "útiles".

5.1. k-means

La primera aproximación que tuvimos fue buscar agrupar nuestro conjunto en clusters que pudieran darnos una pista de cuales eran los fraudes que estaban más diluidos en un conjunto de legítimas. La idea fue entonces:

1. Preprocesar el Train uniformizando las variables para que todas estén en métricas similares.[2]
2. Generar los clusters.
3. Ordenarlos de forma creciente por proporción de fraude.
4. Generar nuevos Train quitando secuencialmente los n primeros clusters (mientras no se quite más del 10 % del fraude).
5. En cada nuevo Train entrenar un clasificador y evaluar.

Dado el tamaño del conjunto de Train, resultó imposible utilizar otro algoritmo que no fuera k-means. Luego de ejecutarlo para que generara 40 clusters se determinó que la cantidad máxima de clusters más diluidos que podían removerse sin quitar más del 10 % del fraude era 16:

Núm de Clusters	Fraudes Removidos	Legítimas removidas
1	0,0 %	0,0 %
4	0,6 %	9,2 %
8	3,4 %	23,4 %
12	6,2 %	33,8 %
16	9,5 %	43,7 %

Tabla 5.1: Cantidad de fraudes y legítimas removidas (en procentaje) al quitar los n clusters más diluidos.

Los resultados de la evaluación por wrapping están plasmados en la gráfica 5.2. Como podemos ver, no existe una diferencia significativa en el desempeño antes y después de realizar cualquiera de estas remociones de clusters. Si bien no conseguimos mejorar los resultados , sí podemos optimizar el tamaño del conjunto de Train haciendo que los modelos sean más sencillos y clasifiquen más rápido. Además, obtuvimos una información relevante: los fraudes que eliminamos podrán no ser "molestos", pero tampoco son "necesarios".

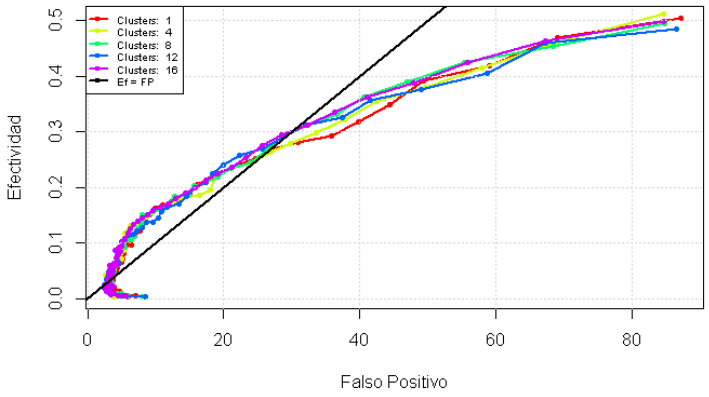

Figura 5.2: Efectividad versus FP para distintas cantidades de clusters removidos.

[2]Recordemos que todas las variables armadas como cocientes fueron forzadas al intervalo [-2,2] al utilizar f, por lo que solo tuvimos que normalizar unas pocas.

Todo este trabajo fue realizado dentro del conjunto de Train completo. Otra posibilidad sería probar de hacer agrupaciones dentro del conjunto que contiene únicamente los fraudes de Train. Como éste es mucho más reducido, pudimos ejecutar un clustering por EM, pero tanto este algoritmo como k-means devolvieron clsuters con cantidades muy similares en cada uno. Esta información no nos permite armar un procedimiento como hicimos para el conjunto de Train original, pues necesitaríamos clusters muy minoritarios en comparación con el resto.

5.2. Local Outlier Factor

La segunda alternativa para detectar cuales podrían ser instancias ruidosas fue utilizar una medida conocida como LOF [3] que da a cada instancia un puntaje de acuerdo a cuan aislada está esta de sus vecinos más cercanos. Cuanto mayor es el número, más alejado está el punto.

El cálculo de esta cantidad está implementado en un filtro de `WEKA`. Al aplicar éste a nuestro conjunto de Train[3] pudimos observar que las transacciones que obtenían puntajes mayores a 5 eran todas legítimas. Además, no parecía haber una diferencia significativa entre los puntajes obtenidos en cada clase, como puede verse en la siguiente tabla :

q	Legítimas removidas	Fraudes Removidos
0,99	$0,99\,\%$	$4,2\,\%$
0,96	$3,97\,\%$	$11,9\,\%$
0,93	$6,96\,\%$	$19,5\,\%$
0,90	$9,95\,\%$	$26,5\,\%$

Tabla 5.2: Cantidad de fraudes y legítimas removidas (en procentaje) al quitar las transacciones cuyo valor de `LOF` sea mayor al cuantil q.

Como puede verse, al remover las transacciones con valores mayores perdemos en proporción muchos más fraudes que legítimas. Esto se debe a que los primeros son muchos menos en cantidad total. Si tenemos en cuenta el desbalance existente (del orden de 1:300 por el submuestreo) es claro que no estamos removiendo una cantidad total tan diferente de cada clase.

La similitud entre las distribuciones del `LOF` en cada clase también es visible en la siguiente gráfica:

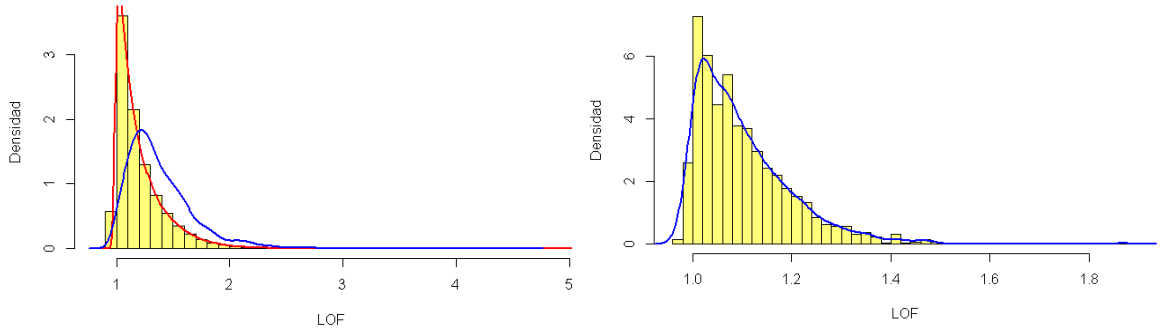

Figura 5.3: Histograma y densidad estimada para la variable `LOF` en los fraudes del conjunto de Train (derecha) y en el conjunto entero de Train (izquierda). En rojo: densidad de las legítimas. En azul: densidad de los fraudes.

De estas observaciones concluimos que no es útil hacer el cálculo del `LOF` sobre el conjunto de Train completo. Sin embargo, al igual que hicimos en el experimento anterior, podríamos probar de hacer el cálculo solo con los fraudes de Train de manera de detectar cuales son los que menos se parecen al

[3]Previa normalización, tal como hicimos en el experimento anterior.

resto . En la gráfica anterior podemos ver la distribución estimada del `LOF` en este conjunto la cual notoriamente no tiene puntos atípicos. Esto indicaría que en la estructura de los fraudes no habría puntos extremadamente dispersos (o que la medida de `LOF` no es la manera adecuada de captarlos). En todo caso, no es de extrañar que al remover sistemáticamente los fraudes con `LOF` mayores a cierto cuantil variable q y hacer la evaluación por wrapping obtengamos que no es de utilidad la remoción de fraudes con este criterio (de hecho, el rendimiento bajó ligeramente en comparación con el obtenido al final de la tercera etapa):

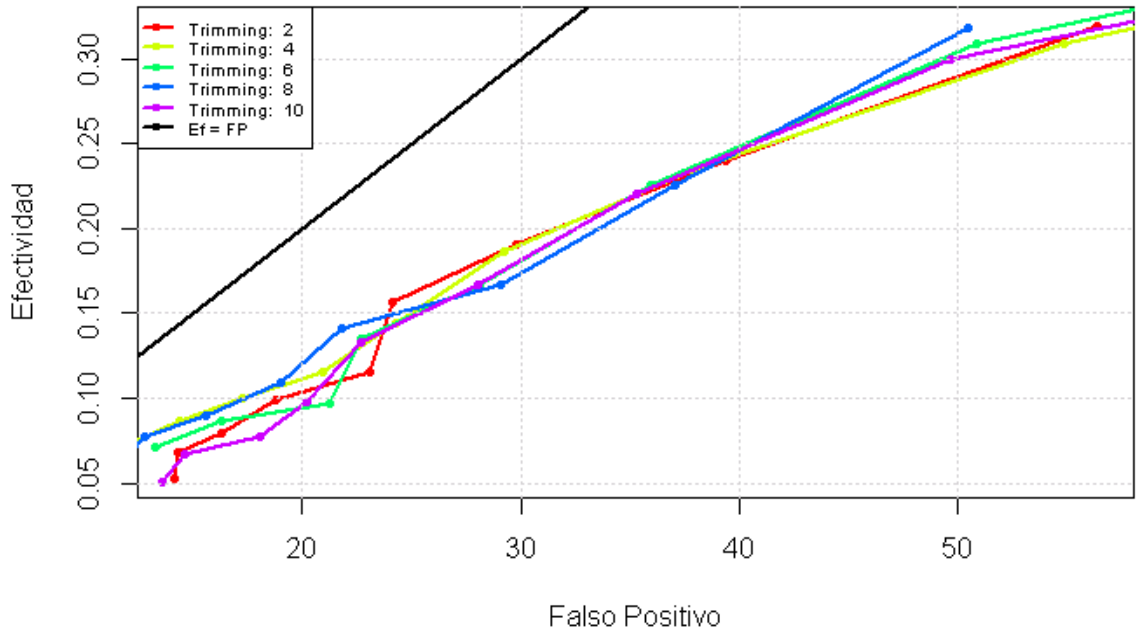

Figura 5.4: Efectividad versus FP al remover el $q\,\%$ de los fraudes con mayor `LOF`.

Aunque en este caso no pudimos hacerlo, recordemos sin embargo que mediante el clustering de k-means parecíamos haber podido detectar fraudes irrelevantes (si bien no molestos). La diferencia sustancial con aquel experimento fue que en esta ocasión solo removimos fruades mientras que antes removíamos fraudes y las legítimas más cercanas. Podemos concluir entonces que: o bien el `LOF` no es una buena manera de detectar outliers en nuestro conjunto de datos, o bien es necesario remover además el vecindario de los fraudes con mayores puntajes. Las pruebas necesarias para dirimir esta cuestión quedarán para una etapa posterior.

Capítulo 6

Conclusiones y trabajo futuro

Después de haber trabajado por un período extenso sobre el mismo problema puedo afirmar que éste no es en absoluto sencillo. Las dificultades que planteamos al principio de este documento son varias y profundas. Sin embargo, otro problema que trae grandes implicaciones y que es mucho más difícil de diagnosticar formalmente (pero que mi familiaridad con el problema me hace intuir que existe) es la mala calidad de los datos. Desde un principio, existen varios campos que naturalmente uno quisiera tener pero que no están disponibles o están corruptos, como ser: asociación de tarjeta con cuenta o con cliente, límites de crédito de las cuentas, fecha de cierre, información de atrasos en pagos, etc. Todas estas variables son intuitivamente muy importantes para armar un perfil del cliente, y sin embargo no contamos con ellas para el problema. Es esperable que si no se cuenta con la información adecuada, por mas potente que sea un algoritmo o técnica, se va a ver en dificultades para extraer valor de donde no lo hay. Como dice una famosa frase: "*Garbarge in, garbage out*".

Sin embargo, no quisiera caer en el error de culpabilizar a un factor externo y quitarme responsabilidades sobre el proceso. Es por esto que aún defendiendo el punto anterior, considero que existe espacio para armar un modelo cuando menos satisfactorio si tomamos decisiones sensatas y creativas. Quisiera por lo tanto dedicar los últimos párrafos de este trabajo a plantear posibles líneas de investigación para continuar avanzando en este problema.

6.1. Análisis del Pipeline

Como hemos descrito anteriormente, el proceso de armado de los modelos se fue dando de forma prolongada en el tiempo y en momentos muy distintos de avance en la materia (y en el área) de Reconocimiento de Patrones. En particular, durante los primeros momentos mi formación en el área era nula, por lo tanto es natural que en retrospectiva haya tomado por caminos que hoy no seguiría.

A continuación presentamos un listado de sugerencias, objeciones u oportunidades que surgen al estudiar el pipeline del armado de modelos:

Limpieza de datos

- Hacer un estudio más profundo de los campos de la base para determinar si no hay variables que puedan ser de utilidad aún cuando tengan muchos valores faltantes. No descartar la idea de realizar cortes con este criterio de modo de entrenar un modelo cuando los datos están y otro cuando no.

Definición de conjuntos de Train, CV y Test

- Revisar el criterio de partición, pues como mencionamos anteriormente hemos comprobado que el fraude no tiene una distribución estacionaria en el año. Luego, realizar la división en meses contiguos del año no tendría ninguna justificación teórica. Se podría probar con alternativas como muestreos completamente aleatorios o estratificados.

- Estudiar si existen alternativas al sobremuestro de `SMOTE`: así como en la etapa final buscamos fraudes ruidosos para sacarlos, podríamos buscar criterios para determinar cuales fraudes son imprescindibles. Luego, podríamos realizar un sobremuestreo únicamente en este subconjunto.

Generación de variables

- Sería interesante probar con transformaciones de las variables originales por si se logra encontrar alguna que aumente el poder descriptivo. En particular, la función f utilizada para los cocientes, si bien ayuda a uniformizar la métrica podría no ser la opción más adecuada.

- Explorar la posibilidad de hacer agrupamiento automático (clustering) en lugar de manual para generar las variables `MCC_Group`, `Amount_Group` y `Time_Range`.

- Así como se creó la variable `Time_Range` para caracterizar el comportamiento de los clientes según el momento del día, podría crearse un nuevo concepto asociado al día o la semana del mes.

Selección de características

- Si bien estamos limitados por los tiempos de procesamiento, el criterio de IG o cualquier otro que evalúe el efecto de las variables por si solas es altamente subóptimo, pues como conocemos de la teoría una variable puede mejorar drásticamente su poder descriptivo en combinación con otra. Sería pues interesante revisar la manera en que se rankean las variables.

- En la misma línea de lo anterior, alta correlación entre variables (o incluso alta información mutua) no implica redundancia necesariamente. Mientras se siga haciendo un wrapping para verificar cada decisión, no vamos a cometer un error. Sin embargo, si podemos estar siendo mal guiados si usamos este criterio.

Algoritmos

- El software `WEKA` si bien es comodo y sencillo de usar no necesariamente es la mejor opción dada la dificultad para ajustar los algoritmos fuera de lo planeado por el programa. Por lo tanto, sería una buena idea buscar una herramienta más flexible. En particular la implementación de RF de `WEKA` solo realiza un trimming muy primitivo que es fijar un valor máximo posible para la profundidad de cada árbol. Dada la gran variedad de criterios posibles que existen para la poda, un algoritmo que permita modificaciones abre un mundo nuevo de posibles ajustes al clasificador.

- La selección de parámetros óptimos de RF se realizó mediante un algoritmo ávido pues se oprimizó de a uno por vez sin volver a revisar las decisiones tomadas. Sería interesante buscar alternativas que al menos se aproximen (si es que son demasiado costosas) a una búsqueda del tipo GridSearch.

6.2. Continuación del trabajo del curso

Además de lo planteado en la sección anterior, existen algunas posibles líneas que surgen de las ideas que empezamos a explorar sobre el último conjunto de experimento, relacionadas con clustering y que surgen de la necesidad de ganar nuevo conocimiento sobre la estructura de los datos. Estas propuestas no son de realización inmediata pero implican un estudio profundo de la realidad de la base por lo que pueden resultar muy provechosas. Para finalizar el trabajo, presentaremos las mismas:

- Teniendo en cuenta que nuestro esfuerzo estuvo enfocado siempre en describir perfiles de clientes, el algoritmo de clustering podría ayudarnos en esta dirección si lo ejecutamos sobre cada cliente por separado. De esta manera sería sencillo señalar cuales son las desviaciones del comportamiento normal para cada cliente específico.

  La mayor dificultad en esta tarea consiste en el cambio paradigmático que conlleva en cuanto a la puesta en producción, ya que dejaría de ser un modelo unificado que simplemente marca las transacciones en línea yendo contra una tabla maestra para pasar a mirar la información en un registro histórico por cliente. Armar un tal sistema es una tarea no trivial, aunque no imposible y de seguro atractiva si mejorará el poder de clasificación.

- Como manera de obtener un mayor conocimiento de lo que realmente hace el clasificador, sería interesante aislar el conjunto de instancias en las que los algoritmos tienen a equivocarse para poder enfocar en ellos la atención y descubrir que características las hacen confusas. A partir de un análisis tal podrían surgir varias opciones interesantes como combinar modelos o determinar cortes nuevos.

Bibliografía

[1] Hastie, Trevor, Tibshirani, Robert and Friedman, Jerome. *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001.

[2] Alexander Vezhnevets and Olga Barinova. 2007. *Avoiding Boosting Overfitting by Removing Confusing Samples*. In *Proceedings of the 18th European conference on Machine Learning* (ECML '07), Joost N. Kok, Jacek Koronacki, Raomon Lopez Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron (Eds.). Springer-Verlag, Berlin, Heidelberg, 430-441.

[3] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J. (2000). *LOF: Identifying Density-based Local Outliers. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD. pp. 93–104. doi:10.1145/335191.335388. ISBN 1-58113-217-4