Predictive Modelling Pycones 2016

Equipo: Lecrín Technologies

Pedro Serrano Prieto (pedrosp@lecrintech.com)
Andrei Corcodel Biboiu (andrei@lecrintech.com)
José Antonio Miñan (jose@lecrintech.com)

Índice

1.Análisis Inicial

Antes de comenzar con el desarrollo de un modelo de clasificación, hicimos algunos análisis iniciales, que nos sirvieran para elegir cuál sería el camino más adecuado para encontrar el modelo más preciso. Cabe destacar que muchas de las tareas, como la generación del set de entrenamiento se ha hecho con **Python**.

Balanceo del problema.

Tras analizar el set de datos *train* descubrimos que el problema un **problema no-balanceado**, con un ratio 1:63, siendo la clase "0" la clase mayoritaria.
Para generar un set de entrenamiento balanceado, para ello hemos separado los registros con clase "1" de los registros con clase "0" y hemos generado un nuevo set de entrenamiento con 7400 registros tomados de forma aleatoria de cada subconjunto, dando un total de 14800 registros.
Posteriormente se probó la técnica **bagging** pero **no mejoró** la precisión del clasificador.

Columna ID

Cuando se hizo la selección de características, se incluyó por error la columna ID, lo cual nos dio un algoritmo con una precisión del 100%. Al analizar el motivo de este curioso acontecimiento, nos dimos cuenta, que las IDs se han asignado de forma autoincremental, primero con los registros con clase "0" y después con los registros de clase "1", por lo tanto, a partir de cierto ID, todos los registros tienen clase "1". Esta curiosidad fue detectada por nuestro árbol clasificador y consiguió una precisión máxima, pero irreal.
Después de esto suprimimos la columna ID y volvimos a realizar la selección de características.

2. Selección de características

Como **preprocesamiento** de datos hemos realizado una selección de características para conseguir las mejores con el objetivo de mejorar el modelo y tiempos de ejecución. Esta parte se ha realizado mediante un algoritmo de **fuerza bruta**, que prueba distintas combinaciones de características y las prueba con un **RandomForest de Weka**.
Se han realizado dos selecciones de características, una **general**, sobre todas las columnas, que ha dado como combinación más efectiva un conjunto de **25 características**; y otra usando únicamente **variables numéricas**, que ha dado como combinación más efectiva un conjunto de **20 características**.

Características generales: TOTAL 25 características

IMP_CONS	IMP_SAL	IND_PROD	IND_TEND	NUM_OPER
3, 12, 15	1, 2, 3, 5, 9, 12, 21	2, 5, 14, 17, 19, 20	4, 5	5, 6, 8, 11, 12, 13, 22

Nota: Los campos de tipo socio_demo, no mejoran los resultados, en cualquier caso los igualan o los empeoran.

Características numéricas: TOTAL 20 características

IMP_CONS	IMP_SAL	NUM_OPER
2, 17	2, 3, 4, 5, 6, 15, 18, 21	2, 4, 5, 7, 13, 14, 18, 19, 21, 22

---

3. Modelos desarrollados

Nuestra estrategia ha sido generar dos modelos de clasificación. El primero, uno simple que **pueda ser visualizado y exportado**. En concreto, se ha usado un árbol de decisión **J48 de Weka**, con la selección de características general (25 columnas) y utilizando el set de entrenamiento, anteriormente descrito de 14800 registros.
Se adjunta ***modelo.txt*** con el árbol de decisión en formato txt, que puede ser importado o visualizado por un humano, para analizar qué variables son más representativas, aportando un potencial **análisis de negocio** futuro.

El segundo modelo, que busca maximizar la precisión es un **multi-view** en el cual se usan las dos selecciones de características que se explicaron anteriormente, la general se procesa con un **RandomForest** y la selección de variables numéricas se procesa con un algoritmo **KernelLogisticRegression** de Weka.
Ambas probabilidades se ensamblan con un nodo de python que genera una probabilidad como media de las probabilidades de ambos algoritmos.

Problema: Este trabajo se inició en nuestras oficinas, en PCs de análisis de datos, pero debido al viaje a la PyCon, tuvimos que terminarlo en ordenadores portátiles de poca potencia, lo cual nos impidió lanzar nuestro modelo multiview ya que el algoritmo KernelLogisticRegression consume demasiados recursos.
Por lo tanto, este modelo no ha podido ser entregado, pero sí se incluye un workflow de Knime que incluye todo el proceso.
Dentro de nuestras posibilidades, hemos hecho un ensemble del algoritmo J48 con un RandomForest, ambos con el set de características generales.

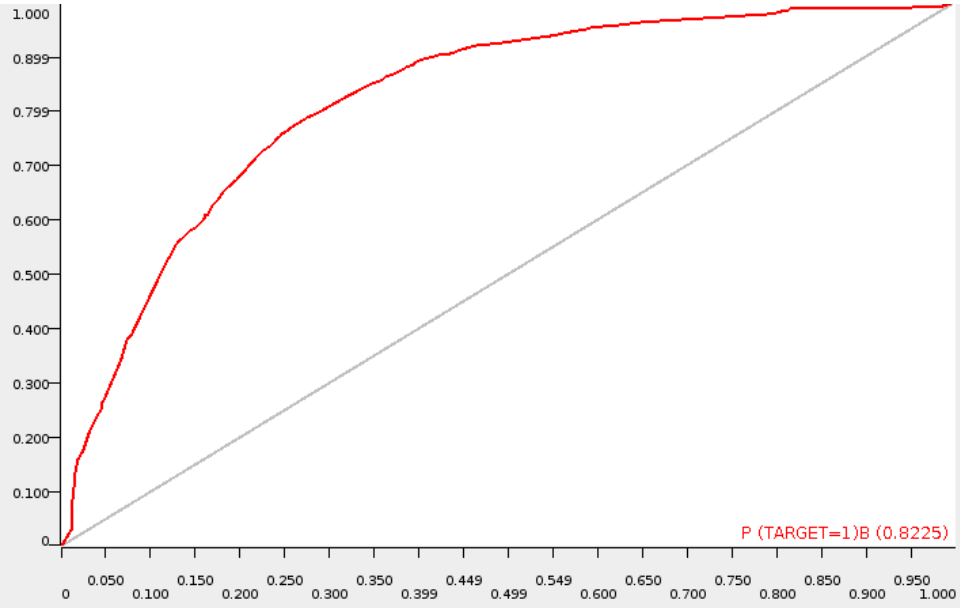
4. Resultados

Para evaluar los resultados y dado que el problema es no balanceado, vamos a utilizar **Área bajo curva ROC (AUC)** como indicativo de la precisión de nuestro modelo.

Modelo 1:

Algoritmo Weka J48

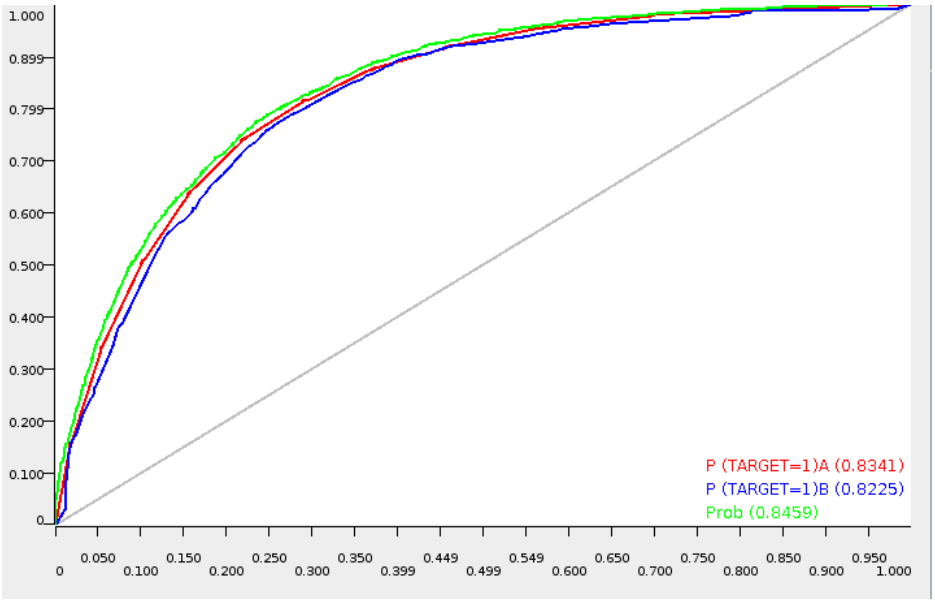
Set de datos 14800 registros balanceado.



Modelo 2:

Ensemble de J48 + Random Forest.

Set de datos 14800 registros balanceados para ambos.



La combinación de ambos algoritmos, mejora la precisión entre 1% y 2%.

Modelo 3:
(No se ha podido reproducir por problemas logísticos)
Multiview RandomForest + KernelLogisticRegression
Set de datos 14800 registros balanceados.
El RF utiliza la selección de características general, el KLR la selección de 20 columnas numéricas.
Gracias al modelo multiview se **puede mejorar el resultado entre un 0.5 y 1.5%** aunque se requiere mayor memoria y mucho más tiempo para clasificar.

En los test realizados en nuestra oficina, este modelo nos dio un **AUC de 0.857.**
Al tener que terminar el proyecto en nuestros portátiles, no hemos podido volver a reproducir el experimento y por lo tanto, no podemos mostrar la curva ROC generada por este modelo.

5. Entregables

1. **test.txt:** Resultados del **Modelo 2** (J48+RandomForest) en el formato pedido para el reto.
2. **modelo.txt:** Modelo en formato texto generado por el J48 de Weka.
3. **WfCajamar.zip:** Workflow de la plataforma libre Knime, sobre el que se ha trabajado para realizar el proyecto. Está construido el Modelo 2, pero se conserva la rama original que utiliza KernelLogisticRegression.