

Memoria de la práctica 2

1. Proceso realizado

Todo el proceso seguido ha sido basado en el taller de Spark en el siguiente repositorio de GitHub:

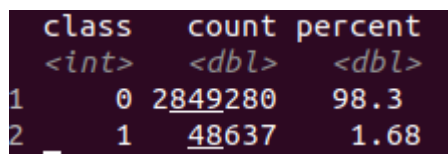
https://github.com/manuparra/taller_SparkR

En primer lugar se han instalado todas las librerías necesarias para ejecutar el código, también se ha indicado la ubicación de las librerías de Spark tal y como se indica en el siguiente repositorio:

<https://github.com/manuparra/TallerH2S>

Leemos los datos con la función `spark_read_csv`, se ha usado el conjunto de datos más grande que se encuentra en la ubicación `hdfs://hadoop-master/user/mp2019/ECBDL14_10tst.data`. Como se indica en el enunciado, nos quedamos con las cinco primeras columnas y la variable de clase gracias a la función `select()`.

Como las clases estaban muy desbalanceadas:



	class	count	percent
	<int>	<dbl>	<dbl>
1	0	2849280	98.3
2	1	48637	1.68

Se ha procedido a hacer un undersampling, para ello se ha identificado la clase mayoritaria y se han escogido muestras aleatorias hasta tener el mismo número que la clase minoritaria. Este proceso ha consistido en crear dos datasets, uno para cada clase y el de la clase minoritaria se ha reducido aleatoriamente. Finalmente se han juntado estos dos datasets en uno.

Una vez tenemos nuestro dataset preparado para el aprendizaje, es necesario separarlo en dos conjuntos de datos, uno de entrenamiento y otro de test. Para ello se ha usado la función `sdf_random_split()` indicando que queremos un 80% de datos de entrenamiento y un 20% de datos de test. En la siguiente imagen se muestra la cantidad de filas que hay en cada conjunto:

```

> count(partitions$test)
# Source: spark<?> [?? x 1]
      n
  <dbl>
1 19614
> count(partitions$training)
# Source: spark<?> [?? x 1]
      n
  <dbl>
1 77038

```

Con el conjunto de datos de entrenamiento se procede a realizar el aprendizaje usando distintos modelos. Los modelos elegidos han sido regresión logística, random forest y decision tree. Se han realizado dos aprendizajes para cada modelo variando algunos de sus parámetros.

Para el modelo de regresión logística se ha realizado un aprendizaje con 100 iteraciones y otro con 200. Para el modelo de random forest se ha realizado un aprendizaje con 20 árboles y otro con 30. Por último, para el modelo de decision tree se ha realizado un aprendizaje con una profundidad de 5 y otro con una profundidad de 8.

En total se han obtenido 6 modelos que se han guardado en una lista para su posterior comprobación de precisión. Para comprobar esta precisión se ha realizado una función que realiza una predicción sobre el conjunto de test para un modelo y devuelve su porcentaje de acierto.

Por último lanzamos la función para todos los modelos para obtener los resultados.

2. Resultados

Tras ejecutar la función comentada anteriormente obtenemos los siguientes resultados.

```

> ml_score
$`Logistic v1`
[1] 0.5312022

$`Logistic v2`
[1] 0.5312022

$`Decision tree v1`
[1] 0.5757112

$`Decision tree v2`
[1] 0.5757112

$`Random Forest v1`
[1] 0.5833078

$`Random Forest v2`
[1] 0.5833078

```

Los resultados vienen en un rango de $[0,1]$ donde 0 significa que ha fallado todas las predicciones y 1 que las ha acertado todas.

3. Conclusiones

Nos hemos encontrado con un problema de desbalanceo de clases muy grande y la única solución encontrada para Spark ha sido realizar undersampling tal y como se vio en clase. Al realizar esto se ha perdido gran cantidad de información que es muy útil para aumentar la precisión de los modelos de aprendizaje.

En cuanto a los resultados se podría decir que son bastante pobres, en todos los casos hemos obtenido porcentajes de acierto entre el 50% y 60%. Estos resultados han sido sobre un conjunto de datos balanceado lo que no está muy mal. Sin embargo, si tenemos en cuenta que el conjunto de datos original tenía una 98% de elementos pertenecientes a una clase, los resultados son desastrosos. Esto se debe a que no se debe obtener un porcentaje de acierto menor que el de la clase mayoritaria porque simplemente un predictor que predijera siempre la clase mayoritaria obtendría un porcentaje de acierto del 98%.

A la hora de decidir qué algoritmo es mejor se observa que el random forest es el que ha obtenido mejores resultado con un 58% de precisión. Por otro lado he observado que la precisión no varía nada al cambiar los parámetros de los modelos. La única explicación que encuentro es que encuentra la mejor solución posible y se queda estancada ahí independientemente de que le digamos, por ejemplo, que haga más iteraciones.