Universidad de Granada

Cloud Computing

Procesamiento y minería de datos en Big Data con Spark sobre plataformas cloud

Curso 2019-2020

Realizado por:

Francisco Solano López Rodríguez
20100444P fransol0728@correo.ugr.es

Máster universitario en Ingeniería Informática

1. Introducción

En los últimos años hemos podido ver un incremento exponencial en la generación de datos, cada día se generan más y más datos. El problema que surge de esto se encuentra en que al trabajar con volúmenes de datos muy grandes, puede ocurrir que el procesamiento en simples ordenadores no sea posible. Sin embargo, el uso de soluciones basadas en el paradigma Cloud Computing, puede ayudarnos a dar soporte a estas necesidades.

En esta práctica vamos a realizar tareas propias de la ciencia de datos, pero esta vez trabajaremos con un conjunto de datos más grande del que solemos estar acostumbrados a tratar. Este conjunto de datos tiene un tamaño de unos 4GB, que a pesar de ser más reducido del que podría estar procesando por ejemplo una empresa dedicada a estas tareas, ya vemos que en una máquina local nos faltan recursos para procesar semejante cantidad de datos en un tiempo razonable.

Para realizar la práctica vamos a trabajar en un servidor de Hadoop, en el que utilizaremos HDFS como sistema de archivos distribuido y el framework Apache Spark para el procesamiento de datos distribuido.

2. Resolución del problema de clasificación

2.1. Filtrado del conjunto de datos

En primer lugar he leído el fichero con la cabecera de los datos y he obtenido el nombre de las columnas. Después he leído el fichero de datos y le he añadido los nombres de las columnas obtenidos anteriormente, para después seleccionar solamente las 6 columnas con las que voy a trabajar. Por último he guardado el nuevo dataset con el nombre de filteredC.small.training. El código asociado a esta tarea se puede encontrar en el siguiente enlace: filterDataset.py.

2.2. Preprocesamiento del conjunto de datos

Una de las columnas que me fueron asignadas para hacer la práctica, era categórica y los valores posibles que podía tener este atributo eran H, E o C, debido a ello convertí la columna a valores enteros. Para hacer esto recurrí a la clase **StringIndexer** que podemos encontrar en pyspark. El código usado para esta función puede consultarse en este enlace: categorical_column_to_int.

Otro problema que nos encontramos en el dataset es que la clase está muy desbalanceada, donde 1375458 filas están etiquetadas como clase 0, mientras que de la clase 1 tan solo hay 687729. Para solucionar esto y balancear las clases, podemos recurrir por ejemplo al submuestreo o al sobremuestro. Ambos algoritmos, tanto undersampling, como oversampling han sido implementados, sin embargo solo se ha hecho uso del submuestreo. Tras aplicar el submuestreo las filas de la clase 1 se han

mantenido y las de la clase 0 se han reducido a 686415 filas. Se puede consultar el código en el siguiente enlace: undersampling and oversampling.

También se han escalado los datos, pero debido a que la clase StandardScaler solamente escala una columna por vez, para facilitar el proceso se ha hecho previamente uso de la clase VectorAssembler, que nos combina las columnas de datos en una sola columna vectorial. VectorAssembler además de ser útil para facilitar el uso de StandarScaler, nos va a venir bien posteriormente para entrenar los modelos de aprendizaje automático.

Una vez obtenida la columna con las características en forma vectorial se han separado los datos en train y test de forma estratificada, es decir manteniendo la proporción de clases en cada conjunto. Se ha dejado un 70 % de los datos para el conjunto de entrenamiento y el 30 % restante para el test. El código de esta función se puede consultar en el siguiente enlace: stratified_train_test_split.

Una vez separado el conjunto de datos en train y test se han escalado los datos, haciendo uso de la clase StandardScaler que escala los datos de forma que estos tengan media 0 y desviación típica igual a 1. Para entrenar el modelo de StandarScaler se ha hecho uso del conjunto de entrenamiento y una vez obtenido el modelo se han transformado los conjuntos train y test. Código: scale_dataset.

2.2.1. Modelos de clasificación

Los modelos de clasificación que se han utilizado han sido: regresión logística, árboles de decisión y random forest. Para cada uno de los algoritmos usados, se han probado varias parametrizaciones.

Logistic Regression

Se han probado 3 parametrizaciones distintas para la regresión logística. Se han variado los parámetros siguientes:

- **maxIter**: sirve para definir el número máximo de iteraciones.
- **regParam**: es un parámetro de regularización.

Experimento	maxIter	regParam
Experimento 1	10	0,1
Experimento 2	15	0,1
Experimento 3	20	0,01

Métrica	Experimento 1	Experimento 2	Experimento 3
Accuracy	0,5302	0,5302	0,53017
Recall	0,5078	0,5078	0,5086
Specificity	0,5129	0,5129	0,5129
F-Score	0,5276	0,5276	0,5280
Auc	0,5363	0,5363	0,5363

Decision Tree Classifier

Al igual que en la regresión logística, para los árboles de decisión también se han probado 3 parametrizaciones diferentes. Los parámetros que se han variado son:

- **maxDepth**: se utiliza para definir la profundidad máxima del árbol.
- **impurity**: es una medida de la homogeneidad de las etiquetas en el nodo.

Experimento	maxDepth	impurity
Experimento 1	10	gini
Experimento 2	10	entropy
Experimento 3	15	gini

Métrica	Experimento 1	Experimento 2	Experimento 3
Accuracy	0,5334	0,5341	0,5141
Recall	0,5841	0,5848	0,5414
Specificity	0,5187	0,5196	0,4973
F-Score	0,5640	0,5647	0,5352
Auc	0,4808	0,4826	0,4846

Random Forest Classifier

El último algoritmo utilizado ha sido random forest y al igual que con los anteriores ha sido probado con 3 parametrizaciones distintas. Uno de los parámetros que se han variado es maxDepth, que ya hemos visto en los árboles de decisión y el otro parámetro que se ha variado ha sido numTrees, con el cual podemos especificar el número de árboles que se van a utilizar.

Experimento	numTrees	maxDepth
Experimento 1	10	10
Experimento 2	20	10
Experimento 3	20	15

Métrica	**Experimento 1**	**Experimento 2**	**Experimento 3**
Accuracy	0,5367	0,5376	0,5205
Recall	0,5741	0,5769	0,5696
Specificity	0,5218	0,5631	0,5043
F-Score	0,5615	0,5631	0,5511
Auc	0,5498	0,5511	0,5286

2.3. Conclusiones

Los datos obtenidos en todos los algoritmos aplicados han sido muy mediocres, probablemente esto sea debido a la selección de las columnas que me fueron asignadas, las cuales seguramente no tendrán un poder predictivo muy alto. Además de esto el servidor dejó de funcionar en los últimos días y finalmente el último día previo a la entrega de la práctica se pudo usar de nuevo. A pesar de ello el servidor estaba bastante saturado en el último día e iba algo lento, lo cual me obligó a reducir el valor de los parámetros de los algoritmos, en particular el número de árboles del clasificador random forest, que creo que habría mejorado levemente los valores de las métricas utilizadas. También para reducir el tiempo de ejecución de los algoritmos, se usé submuestreo, en lugar de sobremuestreo, con lo cual perdí información.

Observando las tablas correspondientes a las métricas de los diferentes clasificadores utilizados, vemos que los mejores resultados han sido los obtenidos por el clasificador random forest en el experimento 2, donde se utilizó un valor de numTrees igual a 20 y una profundidad máxima de los árboles de 10.

Aunque los resultados han sido mediocres, el desarrollo de esta práctica me ha permitido aprender el uso de Spark y ver como a pesar de haber trabajado con un conjunto de datos de 4GB, con el que no habría podido trabajar en mi propio ordenador, haber hecho uso de este framework de computación distribuida en un servidor hadoop, ha hecho que sea posible procesar este conjunto de datos y realizar tareas de ciencia de datos en unos tiempos razonables.