MÁSTER EN INGENIERÍA INFORMÁTICA

Cloud Computing: Servicios y Aplicaciones

Practica 04 : Hadoop - Spark (BigData)

Autores

Hamada Bouhacida
177339003
bouhcidahamada@correo.ugr.es
hamadabouhcida34@gmail.com

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

---

Granada,Septiembre de 2021

Introduction :

The objective of this practice is to learn to process large amounts of data and extract knowledge through the map reduce paradigm. For this, a workflow has been carried out in Spark with which the ECBDL14_IR2 dataset has been analyzed. In addition, using Spark's MLLib library, multiple predictions have been made about it.

Classification problem resolution :

This process has been divided into three main parts:

Data cleaning:

Since the dataset has more than 600 columns, the first step consists of eliminating most of them and keeping only a few previously selected, which in my case are: PredSS_r2_4, PredSS_r2_3, PSSM_r1_-4_F, PredSS_r2, PSSM_r2_-3_A, PSSM_r1_4_W and class. For this I have created a first workflow in spark P4 preprocessed.py.

This flow loads both ECBDL14_IR2.header and ECBDL14_IR2.data into RDDs. From the RDD of ECBDL14_IR2.header it extracts the list of columns and uses it to find the index of each of the columns that have been assigned to me. It then filters the RDD of ECBDL14_IR2.data based on those indicies and finally saves the columns in CSV format to a file.

This flow has been executed in hadoop.ugr.es.

Preprocessed for MLLib:

I have started by loading the csv generated in the previous section into databricks. The sample code of databricks to load CSVs generates a dataframe of strings, this is problematic because half of the variables are numeric. To fix it I have generated a schema with which I have indicated the data type of each column. Passing this schema to the csv reader function will return a dataframe with the appropriate types.

Below I have balanced the data, since in the class variable the zeros appear twice as many times as the ones. For this I have used a resample function to which passing the desired ratio of zeros and ones among others, returns a dataframe with that proportion.

Later I have indexed the categorical variables using a StringIndexer, since the MLLib algorithms that I have used are not capable of working directly with strings.

In the hope of improving the results, I have used the results of the StringIndexer to create a one-hot vector for each categorical variable.

Finally I have grouped all the predictor variables in a single column called features and I have created the training and test sets with a 70/30 partition. Creating a column that groups the predictor variables appears to be a requirement for the MLLib algorithms to function.

Predictions:

I have made predictions with the following models:

1. Random Forest:

a. The first random forest model is a RandomForestClassifier with 10 trees and has yielded an error rate on the test set of 0.409296.

b. The second random forest model has used 50 trees and a maximum depth of 10. The error rate on the test set has been 0.408386, slightly better than the previous case but it has taken a considerable amount of time.

2. Linear Support Vector Machine:

a. The first model has a maximum of 10 iterations and a regParam of 0.1. The error rate on the test set was 0.411708.

b. The second model has a maximum of 20 iterations and a regParam of 0.2. Its results are slightly better than the previous model with an error rate of 0.411706. Although such a small difference is not significant.

3. Decision tree

a. The first model is a decision tree with the default parameters, and has an error rate of 0.411706 over the test set. In the case of this predictor, the predictions made are in floating point, so the predictions column must be converted to 0 or 1, in this case if the predicted value is greater than 0.45 it is rounded to 1.

b. This second model has a maximum depth of 10 and a checkpointInterval of 5. It offers an error rate of 0.413644, so these modifications to the default behavior have worsened the result of the algorithm.

4. Gradient-boosted tree

a. The first model is a GBTRegressor with a maximum of 10 iterations. It has an error rate of 0.408644, one of the best so far. In the same way as in the previous algorithm, the results of this predictor are in floating point, therefore they must be rounded before calculating the error rate.

b. In this second model I had initially tested a maximum iterations of 20 and a maximum depth of 10, but the runtime skyrocketed in such a way that I had to cut it. Finally I decided to put a maximum depth of 3 instead of 10. With this change the execution time was similar to the previous model, although the results were a bit worse, I got an error rate of 0.409537

5. Binomial logistic regression

a. This first model uses a regParam of 0.3, an elasticNetParam of 0.8, maximum iterations of 10, and a binomial family. The error rate is 0.501521, the worst result yet.

b. This second model uses a regParam of 0.2, an elasticNetParam of 0.7, maximum iterations of 20, and a binomial family. The error rate is back to 0.501521, which makes me think that both this model and the previous one are continually predicting the same class.

I have also tried the Naive Bayes and Multilayer perceptron classifier models but for some unknown reason they caused an exception to be thrown in Spark's internal code.

Conclusion :

Spark has seemed like a very interesting tool, but the platform on which I have done most of the work, databricks, is somewhat limited in its free version, so the process has been slow and a bit tedious. Also, as I mentioned before, I have found errors in Spark itself that have prevented me from trying the multilayer perceptron and the naive bayes. Regarding the rest of the predictors, it has caught my attention that all of them, except for Binomial logistic regression, have an error rate of around 40%, a somewhat high rate. I don't know if this is due to limitations of the dataset or to some mistake that I made during the development of this practice.