CENG 790: Big Data Analytics

Assignment 3: Random Forest Classifier

This assignment is part of the evaluation of CENG 790 course and it should be done individually. The deadline is 09 January 2022 23:59. The late submissions will receive a penalty of %25 for each day, starting from 10 January 2022 00:00.

You should submit a report containing the answers of the questions for each part. You may provide the screenshots of the output where necessary. You should also submit the code you produced to answer the questions. Make sure that your code is understandable, use comments extensively to explain what code piece is written for which question.

Make sure you follow the METU Academic Integrity guide. Copying code from internet is also considered as cheating. Make sure the code you produced is original.

Random Forest is a popular ensemble learning method for Classification and regression. The algorithm builds a model consisting of multiple decision trees, based on different subsets of data at the training stage. Predictions are made by combining the output from all of the trees which reduces the variance, and improves the predictive accuracy. For Random Forest Classification each tree's prediction is counted as a vote for one class. The label is predicted to be the class which receives the most votes.

Dataset

The dataset is a credit risk dataset which is provided as part of this assignment in a csv. The code for reading the dataset is also provided in Credit.scala file. If you need more detailed information about the dataset it can be found at https://archive.ics.uci.edu/ml/datasets/Statlog+(German+Credit+Data)

Extract Features

To build a classifier model, you first extract the features that most contribute to the classification. In the credit data set the data is labeled with two classes – 1 (creditable) and 0 (not creditable).

The features for each item consist of the fields shown below:

- Label → creditable: 0 or 1
- Features → {"balance", "duration", "history", "purpose", "amount", "savings", "employment", "instPercent", "sexMarried", "guarantors", "residenceDuration", "assets", "age", "concCredit", "apartment", "credits", "occupation", "dependents", "hasPhone", "foreign"}

In order for the features to be used by a machine learning algorithm, the features are transformed and put into Feature Vectors, which are vectors of numbers representing the value for each feature.

1. Use a **VectorAssembler** to transform and return a new dataframe with all of the feature columns in a vector column.
2. Use a **StringIndexer** to return a Dataframe with the creditability column added as a label
3. Use **randomSplit** function to split the data into three sets: 80% of the data is used to train (and tune) the model, 20% will be used for testing.

Train the model and optimize hyperparameters

4. Next, we train a RandomForest Classifier with the parameters:
    a. **maxDepth**: Maximum depth of a tree. Increasing the depth makes the model more powerful, but deep trees take longer to train.
    b. **maxBins**: Maximum number of bins used for discretizing continuous features and for choosing how to split on features at each node.
    c. **impurity**: Criterion used for information gain calculation
    d. **auto**: Automatically select the number of features to consider for splits at each tree node
    e. **seed**: Use a random seed number, allowing to repeat the results. Use the random seed 1234 in this assignment.

The model is trained by making associations between the input features and the labeled output associated with those features. In order to find the best model, we search for the optimal combinations of the classifier parameters.

You will optimize the model using a pipeline. A pipeline provides a simple way to try out different combinations of parameters, using a process called grid search, where you set up the parameters to test, and MLLib will test all the combinations. **Pipelines** make it easy to tune an entire model building workflow at once, rather than tuning each element in the Pipeline separately. Read more on pipelines: https://spark.apache.org/docs/latest/ml-pipeline.html

Use the **ParamGridBuilder** utility to construct the parameter grid with the following values: maxBins [25, 28, 31], maxDepth, [4, 6, 8], impurity [“entropy", "gini"]

5. Next, you will create and set up a pipeline to make things easier. A Pipeline consists of a sequence of stages, each of which is either an Estimator or a Transformer. For more info read: https://spark.apache.org/docs/latest/ml-tuning.html
   Use TrainValidationSplit that creates a (training, test) dataset pair. It splits the dataset into these two parts using the trainRatio parameter. For example, with trainRatio=0.75, TrainValidationSplit will generate a training and test dataset pair where 75% of the data is used for training and 25% for validation. Use these values in your code. **Note that, this is different from the original random data split.** Here, we further divide the training dataset into training and validation set, for tuning purposes. The final model that has been tuned will be used to evaluate the result on the test set.

The TrainValidationSplit uses an Estimator, a set of ParamMaps, and an Evaluator. Estimatior should be your random forest model, the ParamMaps is the parameter grid that you built in the previous step. The Evaluator should be **new BinaryClassificationEvaluator()**

What are the best combinations for the hyper parameters you optimized?

6. Finally, evaluate the pipeline best-fitted model by comparing test predictions with test labels. You can use **transform** function to get the predictions for test dataset. You can use evaluator’s **evaluate** function to get the metrics. Read more at https://spark.apache.org/docs/latest/ml-classification-regression.html#random-forest-classifier

What are your accuracy values on train and test sets? Feel free to provide more stats and comment on your performance.

HAVE FUN and GOOD LUCK !