Assignment 1

Case Study 1 – Classification and Regression Algorithms

1. For regression, we consider the column "quality" from the dataset as our label as it is a dependent column on all the other columns and the rest of the attributes are considered as features.

   For classification, we are creating a new field in the data source through the code. The field will have the binary value '0' if the quality value is lesser than 6 and '1' if the quality is greater than or equal to 6. The mean of the quality column in 6 and we gave 0 for rows with quality value lesser than the mean.

   Implementation and instructions
   In regular python, the library **Pandas** was used to open the file in python environment. The library function treats the loaded file like a data frame. Then, as a first step of pre-processing the data, normalization was applied to the data using MinMaxScaler function. Followed by that, an anonymous function was created to create a column, with binary values, based on the value of the quality available in each row of the data. For the implementation & testing purposes, the data was segmented into training and test sets (60 & 40 percent correspondingly) using the **randomsplit** function. After the split up, a function gets called to get the model and compute the metrics for that model.

   In Pyspark, the data source was loaded using the spark context and then a filter operation was performed over the source to remove the header information. For the implementation of classification models, a function was created to create a new field, with binary values, based on the quality and the data source was split into training and testing set. Having processed the data source, the implementation of models and evaluation of their performances were done using a function declaration and call to minimize the code complexity. The function takes the model and test data as the input and computes metrics.

   In Scala, we first begin by loading and parsing the file. Then we skip the headers since we do not need to deal with them. We then create "parsedData" which essentially creates a Labeled Point of labels and features. The next step is to divide our data into the test set (40%) and training set (60%).
   With the training set we build models using each of the Regression/ Classification algorithms. With the models created, we then evaluate them on the test data.
   At the end we evaluate the Mean Squared Errors for the Regression algorithms and Training error and Area under ROC for the Classification

algorithms. For Logistic Regression with LBFGS, we have computed the Precision as an evaluation metric as well along with Training Error.

2. Performance metrics on Classification Algorithm
   The performance metrics that we have used to evaluate our Classification algorithms are: Training Error, Area under ROC, Confusion Matrix and Precision. A confusion matrix and accuracy score are used as the metrics to evaluate the various models. The confusion matrix gives out a 2-D matrix with true positive and false positive values. And, the accuracy score metrics gives out the mean accuracy on the given test data. Also, a test error metric was used to evaluate the classification models. It calculates the discrepancy between the actual and estimated value on the test data.

3. Performance metrics on Regression Algorithm
   The performance metrics that we have used to evaluate our Regression algorithms are: Mean Square Error.
   The mean squared error measures the average of the squares of the errors (the discrepancy between the actual and estimated value).
   We picked MSE because we assumed that when the data has high correlation between the columns, MSE seems to be a good way to determine the efficiency of the model.

4. Compare and contrast
   Comparison: We take the example of computing Mean Square Errors for our Regression algorithms. Using regular Python libraries, we get a value of approximately 0.57 and in Apache Spark we get around 3.8 to 4.0. This shows that Python gives us a lower error than Apache Spark. We assume that the discrepancy seems to be large because of the comparatively smaller dataset.

   For classification, we are using the test data and computing the accuracy of the model's prediction by using the **score** function available in the Scipy library. And, in Apache spark, we define a function with the label and features as the input to calculate a score. The approximate score obtained through Scipy is 0.73 and Apache spark is 0.63. For the classification models, there does not seem to be a noticeable difference between the two models.

   Use of Python libraries vs. Use of Apache Spark:
   We would consider using Apache Spark when it comes to dealing with large datasets while simple Python would not prove to be efficient in such cases.

   Note: We have 3 scala files A1(Regression algorithms), A2(Classification-SVMWithSGD & LogisticRegressionWithSGD) and A3(LogisticRegressionWithLBGFS). We used sbt package to get the jar files for all 3. We were able to execute the program with the jar files for A1 & A2 but were unsuccessful with the A3. Hence, we have submitted the .scala files for your reference (We wrote our codes using IntelliJ).

To execute jar file using Spark:
./spark-submit –-class “A1” –master local[2] A1.jar
./spark-submit –-class “A1” –master local[2] A2.jar