BIG DATA ANALYTICS
PROF. DR. MARTIN THEOBALD, ALESSANDRO TEMPERONI & JINGJING XU
SUMMER SEMESTER 2022 – UNIVERSITY OF LUXEMBOURG
Exercise Sheet #2 – Due on April 28, 2022 (before the lecture starts)

Please upload the source code of your solutions on or before the due date to the provided Moodle assignment as a single zip file using your group id as the file name. Provide some brief instructions on how to run your solution to each problem in a file called Problem_X.txt*, and report also the most important results (such as number of results, runtimes, etc.) of your solutions to that problem within this file. Remember that all solutions may be submitted in groups of up to 3 students.*

***Note:* Since this exercise sheet imposes intensive computational workloads for the hyper-parameter tuning and cross-validation steps of your machine learning models, it is highly recommended to run the following experiments on the IRIS HPC cluster by using either the interactive or batching modes of the Spark launcher scripts (see once more the "Getting Started" provided on Moodle for detailed instructions).**

PREDICTING HEART DISEASES VIA DECISION TREES & RANDOM FORESTS **12 points**

Problem 1. Consider the "*Key Indicators of Heart Disease*" dataset as it is available from Kaggle:

`https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease`

You may download the dataset (in CSV format) either from the above URL or via Moodle. This openly available dataset captures 18 variables (9 booleans, 5 strings and 4 decimals) related to possible indicators of heart diseases from about 320K patients in the U.S. The first (boolean) variable captures whether a patient suffers from a *heart disease* (yes/no) and should be used as the target label for the following supervised classification steps. Follow the general methodology provided in `RunDecisionsTrees-shell.scala` to solve the following tasks.

(a) Load the entire CSV file into a corresponding DataFrame either directly via `spark.read.option("inferSchema", true).option("header", true).csv(...)` or via `spark.createDataFrame(rdd).toDF(...)` from an initial RDD within your Spark session. In case you chose the latter option, also explicitly apply a proper schema (see the header of the CSV file) to this DataFrame with meaningful attribute names for this dataset, either by providing the attribute names directly within the `toDF(...)` method or by defining a new `StructType` with respective `StructField` entries to provide the attribute names and their datatypes[1]. **2 points**

(b) In the next step, we wish to include the form of *cross-validations* as they were introduced in the lecture (Chapter 3) in order to more systematically learn the hyper-parameters of a *Decision Tree* classifier. To do so, extend the given shell script by the following steps:

(i) Instead of applying a static 80%/10%/10% split, create a random split of the DataFrame you constructed in (a) into 90% training and 10% testing data. Define a basic *ML Pipeline*[2] consisting of the necessary stages to train a Decision Tree model by using the 90% split as training data.

(ii) Consider the *ML Tuning*[3] guide of Spark to implement a fully automatic hyper-parameter tuning step into your script. Specifically, use your pipeline to initialize a 5-fold cross-validation loop over a fixed hyper-parameter setting (using `impurity="entropy","gini"`, `depth=5,10,15` and `bins=20,50,100` as parameter ranges).

(iii) Instead of the `MulticlassMetrics` package used in the Moodle script, use the `BinaryClassificationMetrics` in your pipeline to measure the average accuracy over the two possible labels of *heart disease* for each of the cross-validation and hyper-parameter iterations. Then fit the model to the training data (caution: this step is expensive!), and finally save the best obtained Decision Tree model into a file.

[1]See: `https://spark.apache.org/docs/latest/sql-programming-guide.html`
[2]See: `https://spark.apache.org/docs/latest/ml-pipeline.html`
[3]See: `https://spark.apache.org/docs/latest/ml-tuning.html`

(iv) Finally, measure *precision*, *recall* and *accuracy* over the two possible labels of *heart disease* for your best ML model and hyper-parameter setting over the 10% test split you created in (i) by using again the `BinaryClassificationMetrics` package of Spark.

(c) Repeat the methodology described in (b) by training a *Random Forest* classifier instead of a single Decision Tree. Investigate the additional range of `numTrees=5,10,20` to also find the *best amount of trees* (using `"automatic"` training mode) in your Random Forest.

Once more, identify the best overall choice of hyper-parameters you obtain from the cross-validations as described in (i)–(iv) and measure the final *accuracy* you obtain for these hyper-parameters with the one that was obtained by the simpler 80%/10%/10% split of the original script. Also here, use the 10% testing set to measure the accuracy of your final model. **2 points**

Report the resulting parameters and the Spark runtimes of this experiment in your `Problem_1.txt` file.

Weather Prediction via Regression Trees & Regression Forests **12 points**

Problem 2. The Integrated Surface Data (ISD) (available at `http://www1.ncdc.noaa.gov/pub/data/noaa/` and from Moodle) of the National Oceanic and Atmospheric Administration (NOAA) provides a rich collection of weather measurements from around the world. The Air Force station in Luxembourg is also included with all its recordings between 1949 and 2022 (these are all files in the subdirectories that start with the USAF identifier `065900`). For a detailed file format and field explanation, read also the specifications provided in `ish-format-document.pdf`.

Note that, also here, you should consider the sample script "`RunDecisionTrees-shell.scala`" provided on Moodle as basis to solve this exercise; the respective NOAA data dumps as well as a parsing function for the particular data format are also provided on Moodle.

(a) Consider the provided Scala script on Moodle for parsing the input files into an appropriate RDD structure in order to then train a *Regression Tree* that can predict the temperature in Luxembourg for a given date and time of the day. Following the instructions from the NOAA specification, we will only need to extract the following fields from the input files

4. `GEOPHYSICAL-POINT-OBSERVATION: date` (split into three fields: `year`, `month` and `day`)
5. `GEOPHYSICAL-POINT-OBSERVATION: time` (extract only the `hour` without the minutes)
7. `GEOPHYSICAL-POINT-OBSERVATION: latitude coordinate`
8. `GEOPHYSICAL-POINT-OBSERVATION: longitude coordinate`
10. `GEOPHYSICAL-POINT-OBSERVATION: elevation dimension`
13. `WIND-OBSERVATION: direction angle`
16. `WIND-OBSERVATION: speed rate`
18. `SKY-CONDITION-OBSERVATION: ceiling height dimension`
22. `VISIBILITY-OBSERVATION: distance dimension`
26. `AIR-TEMPERATURE-OBSERVATION: air temperature`
28. `AIR-TEMPERATURE-OBSERVATION: dew point temperature`

thus considering the `air temperature` as our target attribute. Transform the RDD obtained from the Scala script into a DataFrame (similarly to the instructions provided on Problem 1 of this sheet), and save your DataFrame to a file in order to avoid re-processing and parsing all of the input files every time you run your application. **3 points**

(b) Implement a function to optimize the hyper-parameters and to train a *Random-Forest Regressor* (using `numTrees=10` and `"automatic"` training mode) by splitting the DataFrame you created in (a) into (i) a *training set* with all records from 1949 until 2021 and (ii) a *test set* with all measurements for 2022. You may use the suggested combination of `for` loops provided in the original script or the *ML Pipelines* API of Spark (again see Problem 1) for this step.

Based on your resulting model, predict the temperature for Luxembourg on a few recent dates in 2022, and manually compare your results with the actually measured values (see, e.g., `AccuWeather.com` for some archived recordings of weather data). **3 points**

Hint: See the Spark API for a reference on how to use the regression-based variant of Decision Trees: https://spark.apache.org/docs/latest/mllib-decision-tree.html

(c) Using your previously trained model, predict the temperatures for the Luxembourg weather station along the whole year of 2022, for all days and times at which data is available from NOAA for 2022. Calculate the *Mean Squared Error* (MSE) (see the above URL for an example) between the predicted temperatures and the actual measurements for 2022 (on those dates available from NOAA for 2022). **3 points**

(d) Which of the above attributes do you think has the *highest correlation* with our target attribute, i.e., the `air temperature`?

To answer this question, find compute the *Spearman's correlation coefficient*

$$\rho(X,Y) = 1 - \frac{6\sum_{i=1}^{n}\delta_i^2}{n(n^2-1)}$$

where X is the distribution of observations under the `air temperature` attribute and Y is the distribution of values under each of the remaining attributes (using the entire NOAA dataset for Luxembourg as input).

Note that, in the above formula, $\delta_i := rank(X_i) - rank(Y_i)$ denotes the difference in ranks between two observations X_i and Y_i, while n is total number of observations. You may consider the natural order of values for the categorical attributes to apply this coefficient. You may also want to appropriately discretize the numerical attributes to calculate the coefficient. **3 points**

Hint: you may directly use the implementation of the Spearman coefficient described in https://spark.apache.org/docs/latest/mllib-statistics.html to solve this exercise.

Summarize your results in your `Problem_2.txt` file.

Recommender Systems via Matrix Factorization **12 points**

Problem 3. Consider the AudioScrobbler dataset (available from Moodle) that was introduced in Chapter 3 of the lecture to solve this problem. Also consider the sample shell script "`RunRecommender-shell.scala`" provided on Moodle as basis to solve this exercise. Also here, we will follow up on the idea of extending the provided shell script via hyper-parameter tuning and cross-validations.

(a) Before we can invoke the actual hyper-parameter tuning steps, we need to define a proper quality measure for our resulting recommender engine. To do so, we first need to modify the construction of the training and test sets as follows.

(i) Instead of using the entire dataset to create the `trainData` RDD (including the resolved artist aliases), first create a new RDD, called `trainData100`, by filtering the initial `trainData` RDD, as it is created by the provided Moodle script, such that it contains `Rating` objects of only those users who listened to *at least 100 distinct artists* (these should be 72,748 users in the entire AudioScrobbler dataset).

(ii) Next, create a 90%/10% split of your `trainData100` RDD into two separate `trainData90` and `testData10` RDDs. As before, the 90% split will be used for hyper-parameter tuning, while the 10% split will be used for the final evaluation of your model under the best hyper-parameters you found.

Important: to evaluate the recommendations properly, make sure that, for each user in `trainData100`, you put about 90% of the artists this user has listened to into `trainData90` while the remaining 10% are put into `testData10`. That is, both `trainData90` and `testData10` will contain exactly the same users (but different artists for each of them).

Then, follow the steps provided in the Moodle script to train a first recommender model by using the `trainData90` RDD.

(iii) Next, take 100 random users (similarly to the `someUsers` RDD toward the end of the script) from the `testData10` RDD and compute the top 25 most recommended artists for each of them.

For each of the 100 lists of recommendations, create a new RDD, called `predictionAndLabels` (similarly to the Spark guide[4] and the suggested implementation in the Moodle script) consisting of pairs $(\hat{y}, y)$, where $\hat{y}$ is the recommendation value of the respective artist for the current user, and y is a binary value indicating whether the current user has indeed listened to that artist $(y = 1)$ or not $(y = 0)$ based on the provided `actualArtistsForUser` Scala function.

This RDD should be used as input to the `BinaryClassificationMetrics` API of Spark which then computes the AUC value for each user automatically. Finally, compute the *average AUC* over all of the 100 previously selected users and compare this value also with a similarly computed AUC value when using the `predictMostPopular` baseline recommendation function provided in the script. **6 points**

(b) Next, consider a "manual" search for the best hyper-parameters of your Recommender System. To do so, implement a triple `for` loop (as shown in Chapter 3, Slide 30) in the provided shell script to evaluate your system over the following ranges of hyper-parameters:

- `rank <- Array(10, 25, 50);`
- `lambda <- Array(1.0, 0.01, 0.001);`
- `alpha <- Array(1.0, 10.0, 100.0);`

For each selection of hyper-parameters, compute the AUC measure of your model over the 10% split contained in the `testData10` RDD you created in (a). That is, for each of the 100 random users you selected, check whether they indeed listened to a newly recommended artist and use this information for the AUC computation under each choice of hyper-parameters.

To avoid overfitting, generalize this procedure by applying also a 5-fold cross-validation over `trainData90`. Report the *AUC*, *precision*, *recall* and *accuracy* of your best resulting model and choice of hyper-parameters over the `testData10` split after the cross-validations have finished. Also report the overall runtime in Spark for this experiment. **4 points**

(c) Finally, create a new user profile (with a user id that has not been taken before in AudioScrobbler) and add 10 new ratings of existing artist ids (of your choice) to the original `trainData` RDD created by the "`RunRecommender-shell.scala`" script from Moodle. Using the best choice of hyper-parameters you found in (a)-(b), train a new recommender model with this additional user, print the top 25 recommended new artists for this user, and manually judge the quality of the results.

2 points

Summarize your results in your `Problem_3.txt` file.

[4] See: `https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html`