Exercise on Machine Learning with PySpark

1. Creating a SparkSession

- Import the SparkSession class from pyspark.sql.
- Create a SparkSession object connected to a local cluster. Use all available cores. Name the application 'test'.
- Use the SparkSession object to retrieve the version of Spark running on the cluster.
  Note: The version might be different to the one that's used in the presentation (it gets updated from time to time).
- Shut down the cluster.

2. Loading flights data

- Read data from a CSV file called 'flights.csv'. Assign data types to columns automatically. Deal with missing data.
- How many records are in the data?
- What data types have been assigned to the columns? Do these look correct?

3. Removing Columns and rows

- Remove the flight column.
- Find out how many records have missing values in the delay column.
- Remove records with missing values in the delay column.
- Remove records with missing values in any column and get the number of remaining rows.

4. Column manipulation

- Import a function which will allow you to round a number to a specific number of decimal places.
- Derive a new km column from the mile column, rounding to zero decimal places. One mile is 1.60934 km.
- Remove the mile column.
- Create a label column with a value of 1 indicating the delay was 15 minutes or more and 0 otherwise.

5. Categorical Columns

- Import the appropriate class and create an indexer object to transform the carrier column from a string to a numeric index.
- Prepare the indexer object on the flight data.
- Use the prepared indexer to create the numeric index column.
- Repeat the process for the org column.

6. Assembling Columns

- Import the class which will assemble the predictors.
- Create an assembler object that will allow you to merge the predictors columns into a single column.
- Use the assembler to generate a new consolidated column.

7. Train/test split

- Randomly split the flights data into two sets with 80:20 proportions. For repeatability set a random number seed of 17 for the split.
- Check that the training data has roughly 80% of the records from the original data.

8. Build a Decision Tree

- Import the class for creating a Decision Tree classifier.
- Create a classifier object and fit it to the training data.
- Make predictions for the testing data and take a look at the predictions.

9. Evaluate a Decision tree

- Create a confusion matrix by counting the combinations of label and prediction. Display the result.
- Count the number of True Negatives, True Positives, False Negatives and False Positives.
- Calculate the accuracy.

10. Build a Regression model

- Import the class for creating a Logistic Regression classifier.
- Create a classifier object and train it on the training data.
- Make predictions for the testing data and create a confusion matrix.

11. Evaluate the logistic model

- Find the precision and recall.
- Create a multi-class evaluator and evaluate weighted precision.
- Create a binary evaluator and evaluate AUC using the "areaUnderROC" metric.

12. Encoding flight origin

- Import the one-hot encoder class.
- Create a one-hot encoder instance, naming the output column 'org_dummy'.
- Apply the one-hot encoder to the flights data.
- Generate a summary of the mapping from categorical values to binary encoded dummy variables. Include only unique values and order by org_idx.

13. Flight duration model.

- Create a linear regression object. Specify the name of the label column. Fit it to the training data.
- Make predictions on the testing data.
- Create a regression evaluator object and use it to evaluate RMSE on the testing data.

14. Interpreting the coefficients

- What's the intercept?
- What's the intercept?
- Extract the element from the vector which corresponds to the slope for distance.
- Find the average speed in km per hour.

15. Bucketing departure time

- Create a bucketizer object with bin boundaries which correspond to 0:00, 03:00, 06:00, …, 24:00. Specify input column as `depart` and output column as `depart_bucket`.
- Bucket the departure times. Show the first five values for `depart` and `depart_bucket`.
- Create a one-hot encoder object. Specify output column as `depart_dummy`.
- Train the encoder on the data and then use it to convert the bucketed departure times to dummy variables. Show the first five values for `depart`, `depart_bucket` and `depart_dummy`.

16. Flight duration model: Adding departure time

- Find the RMSE for predictions on the testing data.
- Find the average time spent on the ground for flights departing from OGG between 21:00 and 24:00.
- Find the average time spent on the ground for flights departing from OGG between 00:00 and 03:00.
- Find the average time spent on the ground for flights departing from JFK between 00:00 and 03:00.

17. Flight Duration model: more features

- Fit a linear regression model to the training data.
- Generate predictions for the testing data.
- Calculate the RMSE on the testing data.
- Look at the model coefficients. Are any of them zero?

18. Flight duration model: Regularisation!

- Fit a linear regression model to the training data.
- Calculate the RMSE on the testing data.
- Look at the model coefficients.
- How many of the coefficients are equal to zero?

19. Flight duration model Pipeline stages

- Create an indexer to convert the 'org' column into an indexed column called 'org_idx'.
- Create a one-hot encoder to convert the 'org_idx' and 'dow' columns into dummy variable columns called 'org_dummy' and 'dow_dummy'.
- Create an assembler which will combine the 'km' column with the two dummy variable columns. The output column should be called 'features'.
- Create a linear regression object to predict flight duration.

20. Flight duration model: Pipeline model

- Import the class for creating a pipeline.
- Create a pipeline object and specify the indexer, onehot, assembler and regression stages, in this order.
- Train the pipeline on the training data.
- Make predictions on the testing data.

21. Cross validating simple flight duration model

- Create an empty parameter grid.
- Create objects for building and evaluating a linear regression model. The model should predict the "duration" field.
- Create a cross-validator object. Provide values for the estimator, estimatorParamMaps and evaluator arguments. Choose 5-fold cross validation.
- Train and test the model across multiple folds of the training data.

22. Cross validating flight duration model pipeline

- Create a string indexer. Specify the input and output fields as org and org_idx.
- Create a one-hot encoder. Name the output field org_dummy.
- Assemble the km and org_dummy fields into a single field called features.
- Create a pipeline using the following operations: string indexer, one-hot encoder, assembler and linear regression. Use this to create a cross-validator.

23. Optimizing flights linear regression

- Create a parameter grid builder.
- Add grids for with regression.regParam (values 0.01, 0.1, 1.0, and 10.0) and regression.elasticNetParam (values 0.0, 0.5, and 1.0).

- Build the grid.
- Create a cross validator, specifying five folds.

24. Dissecting the best flight duration model

- Retrieve the best model.
- Look at the stages in the best model.
- Isolate the linear regression stage and extract its parameters.
- Use the best model to generate predictions on the testing data and calculate the RMSE.

25. Delayed flights with Gradient-Boosted Trees

- Import the classes required to create Decision Tree and Gradient-Boosted Tree classifiers.
- Create Decision Tree and Gradient-Boosted Tree classifiers. Train on the training data.
- Create an evaluator and calculate AUC on testing data for both classifiers. Which model performs better?
- For the Gradient-Boosted Tree classifier print the number of trees and the relative importance of features.

26. Delayed flights with random forest

- Create a random forest classifier object.
- Create a parameter grid builder object. Add grid points for the featureSubsetStrategy and maxDepth parameters.
- Create binary classification evaluator.
- Create a cross-validator object, specifying the estimator, parameter grid and evaluator. Choose 5-fold cross validation.

27. Evaluating Random Forest

- Print a list of average AUC metrics across all models in the parameter grid.
- Display the average AUC for the best model. This will be the *largest* AUC in the list.
- Print an explanation of the maxDepth and featureSubsetStrategy parameters for the best model.
- Display the AUC for the best model predictions on the testing data.