iscte TECNOLOGIAS E ARQUITETURA

Algoritmos para Big Data
Lab class handout #4

Machine learning problem – binary classification **2024/2025**

Learning objectives

This lab class is about binary classification in a discrete space. We will setup a ML processing pipeline to achieve the goals, and the data to be considered relates to the domain of banking industry. Specifically, it is a case of fraud detection in credit cards transactions.

Note: It is acceptable that the work to be undertaken may require some time beyond the class. If that is the case, work should continue outside of class. And if needed, assistance will be provided during the office hours established or to be agreed with the lecturer.

Supporting information

- Course slides
- Machine Learning Library (MLlib) Guide
- Apache Spark ML pipeline

Problem to solve

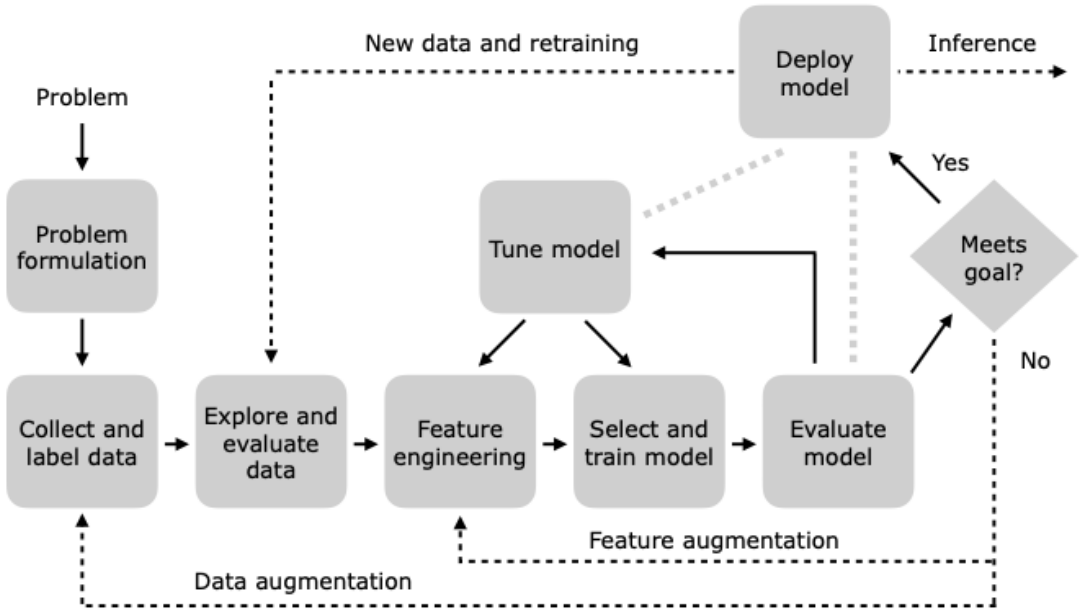
This problem is about credit card fraud detection, aiming to figure out whether a particular credit card transaction is fraudulent or not. Our case-study is based on a Kaggle dataset that holds synthetic data about credit card transactions.

The dataset to be used is in an archive file located at:
`https://bigdata.iscte-iul.eu/datasets/credit-cards-transactions.zip`

Alongside this handout describing the tasks to accomplish, it will be provided a notebook that it is expected to be completed with the implementation of task B.

ML workflow

In order to solve the problem described above, we will setup a ML workflow/pipeline. Recall that a typical ML workflow is designed to work as depicted below:



We will follow the workflow above. On the other hand, we have to consider the practical use of ML pipelines available in Apache Spark. As stated in the documentation: "ML Pipelines provide a uniform set of high-level APIs built on top of DataFrames that help users create and tune practical machine learning pipelines."

Notice that it is possible to combine multiple algorithms into a single pipeline. Besides DataFrames, the implementation involves the following concepts:

1. Transformer: an algorithm which can transform one DataFrame into another DataFrame. For example, an ML model is a Transformer, which transforms a DataFrame with features into a DataFrame with predictions.
2. Estimator: an algorithm which can be fit on a DataFrame to produce a Transformer. For example, a learning algorithm is an Estimator, which is training on a DataFrame and produces a model.
3. Pipeline: the way to chain multiple Transformers and Estimators together in order to specify a ML workflow.
4. Parameter: all Transformers and Estimators share a common API for specifying parameters.

Tasks to accomplish

A. Data ingestion, data preparation and understanding

In this initial task, execute the following operations:

1. Download the data file already mentioned and save it in a proper directory, within the working directory of the Pyspark instalation.

2. Create a notebook specifically for the implementation of this task. In that respect, it is worth looking at previous notebooks that were provided.

3. After creating a *SparkSession* to work with, read the data into a DataFrame and then check it. Make sure that the data is prepared/cleaned to be worked with. For example, check (i) if there are duplicated rows or not, (ii) the existence of NULLs and (iii) if datatypes are properly set or not.

   Hereafter, we will consider the name of the DataFrame containing the data of concern as `df_transactions`.

4. With `df_transactions`, carry out some exploratory data analysis based on descriptive analytics and visualizations. For example:
   - Use the statistical method *describe*() to figure out outliers.
   - Use the method *distinct*() to find unique values in columns of interest.
   - Compute correlations among numerical columns (with no NULLs), and plot them using *plotly.express*.
   - Compute various aggregations on some columns of interest, and plot them using *plotly.express*. Cases to be considered can be:
     - Number of transactions by (i) year and (ii) month.
     - Counting regarding the channel used in transactions (chip usage).
     - Counting of fraudulent transactions versus all transactions, and checking the kind of fraudulent transactions.
     - Counting regarding the channel used in fraudulent transactions (chip usage).
     - Counting and maximum amount regarding fraudulent transactions, by hour.

   If from the analysis above changes on `df_transactions` are warranted, including getting rid of some columns, or creating new columns derived from existing ones, do so.

5. Regardless of changes being made or not, create another dataframe after `df_transactions`, to be named `df_transactions_small`, which will be a 30% sampling (Feel free to use a different sampling fraction, but substantially lower than the maximum).

   Save these two dataframes as parquet files. This concludes the first notebook.

B. ML classifier model

This task will be implemented in the notebook provided. It is assumed that the data regarding `df_transactions_small` mentioned in the previous task has been stored as a parquet file. This is the data that will be used to create the classifier model.

Notice: If for some reason you were not able to generate the clean data in the previous task, there are two parquet files available in the data server that you may use instead. The archive files are: `cards-transactions.zip` and `cards-transactions-small.zip`.

Execute the following operations:

1. After creating a *SparkSession* to work with, read the data into a DataFrame from the parquet file corresponding to `df_transactions_small` already mentioned, and then check it. Hereafter, we will simply call this new DataFrame as `df_clean`.

2. From `df_clean`, compute correlations among numerical features (with no NULLs), and plot them using *plotply.express*.

3. At this point, it is worth remembering the exploratory analysis that has been done in the previous task. Indeed, we are about the select features, whether existing ones or derived, encoding and vector assembling, so the algorithms can use them. It is also important the understanding of data domain/business.

   Once the features for the classifier are set out, we have to enter into the specifics of the algorithms. In that regard, if needed, we will use *StringIndexer*() and *OneHotEncoder*() to move from text to numbers. Also, because algorithms require all input features being contained into a single vector, we need a transformer to do so, in this case *vectorAssembler*().

   This would conclude the feature engineering aspect of the ML workflow.To achieve that:

   (a) Create a new column in `df_clean`, to be called `Fraud`, meaning there is fraud (1.0) if the feature `Is Fraud` is true; otherwise 0.0. This feature will be the label/target of the model.

(b) Set an array containing the names of the features to be used in the model. You may decide which ones are more appropriate at this point in time, although that may change later on as consequence of further analysis and experimentation. Also, from these selected features, set one array with the names of categorical features and another one with the names of non-categorical features.

(c) Create a vector assember with the features mentioned, with the help of *StringIndexer*(), *OneHotEncoder*() and *vectorAssembler*().

4. Now it is time to start the process of selecting and training the model. Let us assume that we have elected the Linear Support Vector Machine algorithm to create the model (see documentation). Hence:

   (a) First, create two dataframes – `df_train` and `df_validation` [1] – after `df_clean`, randomly, with distinct sizes. For instance, use a split of 70%, 30% respectively (or else 80%, 20%). Recall that the model fitness can be affected by the splitting of data (balanced, over-fitting or under-fitting).

   Also, save these dataframes as parquet files. Notice that, as it is a sampling, there is no guarantee of getting the same data split when using the code in a different computer/time. And we may want to reproduce or share the experiments made.

   Notice: It seems data is imbalanced regarding the fraudulent cases reported, which is our feature target – we want to know if a particular transaction is fraudulent or not. This is very common in this type of problem (fortunately, the lower the number of frauds is the better!). But having data imbalancing in the target feature poses challenges to ML model creation. We leave this aspect for a second round of model tuning.

   (b) Just for the purpose of having more memory space hereafter, delete `df_clean` as we no longer need it.

   (c) Set the classifier estimator as *LinearSVC*, with the feature `Fraud` as label/target (mentioned above).

   (d) Set a ML pipeline configuration using the transformer *Pipeline*, with the various stages leading to the model. That is, related to *StringIndexer*(), *OneHotEncoder*(), *vectorAssembler* and *LinearSVC*. Save the pipeline for further use, should it be required.

   (e) Create the model by fitting the pipeline that has been set up to the training data, `df_train`. Save the model for further use, should it be required.

5. Evaluate the model that has been created. To do so:

   (a) Make predictions by applying the verification data, `df_verification`, to the model (which is a transformer).

   (b) Print out the schema of the resulting DataFrame and show the columns:

   `features, rawPrediction, prediction, Fraud`.

   (c) Compute the evaluation metric *areaUnderROC* with the help of the evaluator *BinaryClassificationEvaluator*.

   (d) Compute the confusion matrix and show the values.

   Recall that a confusion matrix is defined by the counting of:

True Positive (TP):	The prediction was positive and it is true.
True Negative (TN):	The prediction was negative and it is true.
False Positive (FP):	The prediction was positive and it is false.
False Negative (FN):	The prediction was negative and it is false.

[1] We use this split and naming on the basis that we would expect to get a separate and unseen sample just for testing, in order to get an unbiased final evaluation of the model that we are about to create.

(e) With the values of the confusion matrix obtained, compute the evaluation metrics: *Accuracy*, *Precision*, *Recall*, *Specificity* and *F1 score*.

Consider that, given the confusion matrix values, the metrics are defined as follows:

- *Accuracy* = (TP + TN) / (TP + TN + FP + FN)
  How often the classifier is correct? (score)
  Metric widely used but not so useful when there are many TN cases.
- *Precision* = TP / (TP + FP)
  Positive predictive value – proportion of positive results that were correctly identified.
  It removes NP and FN from consideration.
- *Recall* = TP / (TP + FN)
  True positive rate. (hit rate, sensitivity)
- *Specificity* = TN / (TN + FP)
  True negative rate. (selectivity)
- *F1 score* (*measure*) = 2 * *Recall* * *Precision* / (*Recall* + *Precision*)
  Useful metric because it is difficult to compare two models with low precision and high recall or vice versa. Indeed, by combining recall and precision it helps to measure them at once.

C. Additional exercise

As an additional exercise, improve the model by taken into account the results that have been obtained so far. It is a tuning process so any decision previously made is open for debate. For example:

- How to interpret the scores obtained?
- How to handle class imbalance, so clear in the data?
- Could the data splitting be enhanced?
- Could a model with different set of features and/or target engineering will perform better?

Once a final and good model has been created, use it with the `df_transactions` data from the previous task, that is, the complete dataset, but excluding the `df_transactions_small` data. The idea now is apply the model only to data that has not contributed to the construction of the model, directly (e.g. `df_train`) or indirectly (e.g. `df_validation`). Evalute the new outcome.