ECE Paris Big Data Ecosystem

Project Report

Henintsoa RAZAFINDRAZAKA, James WANG

31/12/2020

1. Project description

This project combines **Machine Learning** with **distributed systems**, and in particular the computing power distribution. Since the goal of this project is primarily the implementation of a distributed solution, we won't cover in details the implementation of the Machine Learning part. However, it remains interesting to understand the underlying goal of this algorithm.

First, we used a data set on card fraud with the idea of designing a model that could predict all fraudulent credit card transactions. The dataset has been provided by Kaggle on the following link: https://www.kaggle.com/mlg-ulb/creditcardfraud. This is a real-life example of Machine Learning application which allows banks to detect more easily some unusual transactions.

So, our goal would be to predict whether a particular transaction is a fraud or not. This example deals with a classification problem, that's why we have tested the following algorithms in this project: Logistic Regression, Decision Tree Classifier, Random Forest and Gradient Boosted Classifier. We have implemented those algorithms in a distributable manner with **pyspark**, which is the **Python** client for **Spark**. For each algorithm, we have created a python script that allows the creation of a model and its metrics (its performance on the classification task).

Finally, this kind of project may be confusing to the existence of the multiple models there are. That's why it would be convenient to use a versioning tool for these models, and for Machine Learning, we have found the **DVC versioning tool** very useful and practical. Thanks to it, we have been able to control the versions of our data, as well as our models with their respective metrics. This allows us to perform some comparisons to choose the right model.

Now that we have a clear understanding of the project and its main stages, let's dive into the implementation steps of our project.

2. Implementation steps

A. Python scripts with pyspark

First, we needed to begin with the implementation of the different Python scripts, corresponding to the different algorithms quoted before. So, as we have 4 algorithms, we will have 4 Python scripts, each one producing its own model, with its own metrics.

Naturally, all the scripts begin with the necessary imports and the **SparkSession** creation. Though each script being different, the beginning of each one of them consists of the loading of the data, followed by a short preprocessing step.

```
11  spark = SparkSession.builder.getOrCreate()
12
13  loc = os.path.abspath("")
14  data_loc = f"{loc}/data/creditcard.csv"
```

All the columns being already numerical, it needed no other transformation to be ready. The only thing to do was to assemble all the input columns / features into only one column, so that the resulting dataframe would only have 2 columns, the one with the assembled features, and the one with the labels. All these steps are common to all of the algorithms.

```
columns = [x for x in df.columns if x != 'Class']

vector_assembler = VectorAssembler(
    inputCols=columns, outputCol="features"
)
```

After the preprocessing, all the algorithms follow the same principle except the Logistic Regression one because of some particularities for that model. The main difference is that for the 3 other models, we can include in the pipeline the algorithm itself along with the preprocessing step, whereas with the Logistic Regression, we cannot add the algorithm into the pipeline. Anyway, that's just an implementation detail.

For the Decision Tree classifier for example, here is how a pipeline is constructed:

```
DT = DecisionTreeClassifier(labelCol="Class", featuresCol="features")

pipelineDT = Pipeline(stages=[vector_assembler, DT])
```

After that, we simply fit the data into the pipeline for training, then compute the predictions with the test set for testing.

```
modelDT = pipelineDT.fit(train)

predictionsDT = modelDT.transform(test)
```

Finally, we just extract the metrics (accuracy, precision, recall, f1) to evaluate the performance of the models and create a json file to record these. The last step of the script saves the created model into a Parquet format in a specified folder.

B. Data version control with DVC

The Python scripts we have covered before are not run on their own but with DVC with some configurations. Firstly, to use DVC, we need to initialize it in the git repository of the project by running the following command:

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc init

You can now commit the changes to git.

+---------------------------------------------------------------------+
|                                                                     |
|        DVC has enabled anonymous aggregate usage analytics.         |
|     Read the analytics documentation (and how to opt-out) here:     |
|             <https://dvc.org/doc/user-guide/analytics>              |
|                                                                     |
+---------------------------------------------------------------------+

What's next?
------------
- Check out the documentation: <https://dvc.org/doc>
- Get help and share ideas: <https://dvc.org/chat>
- Star us on GitHub: <https://github.com/iterative/dvc>
```

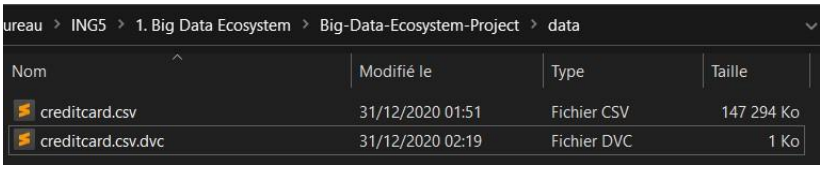
Then we perform a simple git commit to keep track of the initialization and the created files.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> git commit -m "
Initialize DVC"
[main e1b427b] Initialize DVC
 8 files changed, 320 insertions(+)
 create mode 100644 .dvc/.gitignore
 create mode 100644 .dvc/config
 create mode 100644 .dvc/plots/confusion.json
 create mode 100644 .dvc/plots/confusion_normalized.json
 create mode 100644 .dvc/plots/default.json
 create mode 100644 .dvc/plots/scatter.json
 create mode 100644 .dvc/plots/smooth.json
 create mode 100644 .dvcignore
```

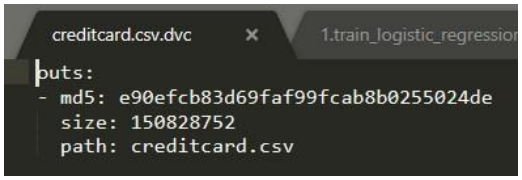
Now that DVC is ready to use, we will begin by versioning the data. This can be done by issuing a “dvc add” command.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc add .\data\
creditcard.csv
100% Add|                                                                    |1/1 [00:00,  1.25file/s]
```

This creates a .dvc file located at the same directory as the data itself.



This newly created file represents a kind of pointer that records the version of the data and can be checked to keep track of the version of it and its location.



Now, DVC is the source code manager of the data and the data itself won’t be uploaded on github. Only the .dvc file that serves as a pointer is going to be pushed on github, that is quite useful when we deal with large files.

```
Henintsoa@DESKTOP-U8BBOJI MINGW64 ~/Desktop/ING5/1. Big Data Ecosystem/Big-Data-
Ecosystem-Project (main)
$ git commit -m "Add raw data"
[main d4414b8] Add raw data
 1 file changed, 4 insertions(+)
 create mode 100644 data/creditcard.csv.dvc
```

Another interesting feature of DVC is the remote storage. Indeed, we can configure a remote storage to which we would be able to push or pull the data or the models. The remote storage can be a cloud solution such as azure or AWS, as well as a big data platform such as Hadoop with HDFS. Because we don’t have access to any Hadoop server for this project, we have chosen Google Drive for the sake of the example.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc remote add
-d remoteProject gdrive://1Y48GIUlQNEmawuT9zWSrNB17lfkFVnf4
Setting 'remoteproject' as a default remote.
```

Some modifications have occurred in the configuration file of DVC, following this command.

```
Henintsoa@DESKTOP-U8BBOJI MINGW64 ~/Desktop/ING5/1. Big Data Ecosystem/Big-Data-
Ecosystem-Project (main)
$ git commit .dvc/config -m "Configure remote storage"
[main 132f5d4] Configure remote storage
 1 file changed, 4 insertions(+)
```

Having configured the remote storage, we can now push our data to it by issuing a “dvc push” command.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc push
  0% Querying cache in gdrive://1Y48GIUlQNEmawuT9zWSrNB17lfkFVnf4|   |0/1 [00:00<?,     ?file/s]G
o to the following link in your browser:

    https://accounts.google.com/o/oauth2/auth?client_id=710796635688-iivsgbgsb6uv1fap6635dhvuei0
9o66c.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=https%3A
%2F%2Fwww.googleapis.com%2Fauth%2Fdrive+https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.appdata&
access_type=offline&response_type=code&approval_prompt=force

Enter verification code: 4/1AY0e-g4zuuUG1RFTcOdUey0KvXWabYcCVonWLy0PJgvsMzUytzYRjj0ciSk
Authentication successful.
1 file pushed
```

Now it’s time to run our previously written scripts which, as recall, creates a model given our data, along with some metrics records. To run our script with DVC configuration, we use the “dvc run” command as follows:

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc run -n train -f `
>> -d data/creditcard.csv `
>> -M scores/metrics.json `
>> py .\src\1.train_logistic_regression.py
```

This command is used to create and add a stage named “train” to the pipeline. The “-n” parameter specifies the name of the stage, the “-f” allows to overwrite the stage if it’s an existing one, the “–d” specifies a dependency to the stage, while the “-M” specifies the metrics linked to the model. The last part corresponds to the command to execute.

The saved model with pyspark is saved in the Parquet format, which explains why the resulting model is a directory. To track the model with DVC, we use the “dvc add” command again, specifying the directory corresponding to the model. It creates another .dvc file, as said before a kind of pointer that allows to track the actual model.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc add .\models\LR_model\

100% Add|                                                                    |1/1 [00:00,  2.35file/s]
```

Finally, we just need to commit and tag the model to perform a proper versioning of it.

```
Henintsoa@DESKTOP-U8BBOJI MINGW64 ~/Desktop/ING5/1. Big Data Ecosystem/Big-Data-Eco
system-Project (main)
$ git add .

Henintsoa@DESKTOP-U8BBOJI MINGW64 ~/Desktop/ING5/1. Big Data Ecosystem/Big-Data-Eco
system-Project (main)
$ git commit -m "First model, Logistic Regression"
[main 593b02c] First model, Logistic Regression
 6 files changed, 30 insertions(+), 5 deletions(-)
 create mode 100644 dvc.lock
 create mode 100644 dvc.yaml
 create mode 100644 models/.gitignore
 create mode 100644 models/LR_model.dvc
 create mode 100644 scores/.gitignore

Henintsoa@DESKTOP-U8BBOJI MINGW64 ~/Desktop/ING5/1. Big Data Ecosystem/Big-Data-Eco
system-Project (main)
$ git tag -a "v1.0" -m "model v1.0, LogReg"
```

We repeat the same procedure for each algorithm, which gives us 4 different versions. Each time we run the “dvc run” command, a dvc.yaml configuration file is created. Here an example with the Gradient Boosted algorithm :

Having created all the versions with the different algorithms, we can now compare their metrics by using the “dvc metrics diff” command as follows:

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc metrics diff v1.0 v2.0
 --targets scores/metrics.json scores/metrics.json
Path                  Metric     Old      New      Change
scores\metrics.json   accuracy   0.99937  0.99935  -2e-05
scores\metrics.json   f1         0.99933  0.99931  -2e-05
scores\metrics.json   precision  0.99934  0.99932  -2e-05
scores\metrics.json   recall     0.99937  0.99935  -2e-05
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc metrics diff v2.0 v3.0
 --targets scores/metrics.json scores/metrics.json
Path                  Metric     Old      New      Change
scores\metrics.json   accuracy   0.99935  0.99944  9e-05
scores\metrics.json   f1         0.99931  0.99942  0.00011
scores\metrics.json   precision  0.99932  0.99941  0.0001
scores\metrics.json   recall     0.99935  0.99944  9e-05
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc metrics diff v3.0 v4.0
 --targets scores/metrics.json scores/metrics.json
Path                  Metric     Old      New      Change
scores\metrics.json   accuracy   0.99944  0.99942  -2e-05
scores\metrics.json   f1         0.99942  0.99939  -3e-05
scores\metrics.json   precision  0.99941  0.99939  -2e-05
scores\metrics.json   recall     0.99944  0.99942  -2e-05
```

Eventually, DVC allows us to visualize a DAG (Directed Acyclic Graph) of stages, which is useful if we want to picture the dependencies and the outputs of each stages easily. In our case, we only have one stage, that’s why our DAG is that simple.

```
PS C:\Users\henin\Desktop\ING5\1. Big Data Ecosystem\Big-Data-Ecosystem-Project> dvc dag
WARNING: Unable to find `less` in the PATH. Check out <https://man.dvc.org/pipeline/show> for more info.
+-------------------------+
| data\creditcard.csv.dvc |
+-------------------------+
              *
              *
              *
         +-------+
         | train |
         +-------+
+----------------------+
| models\DT_model.dvc |
+----------------------+
+----------------------+
| models\GB_model.dvc |
+----------------------+
+----------------------+
| models\LR_model.dvc |
+----------------------+
+----------------------+
| models\RF_model.dvc |
+----------------------+
```

Every model has been versioned, we can move from a version to another by using the “git checkout” followed by “dvc checkout”.

3. Problems encountered

Before using mllib with pyspark, we have tried multiple solutions.

First, we have attempted to use Elephas with the Deep Learning framework Keras. However, with Keras we had some dependency issues, because Keras has moved into Tensorflow whereas Elephas is entirely dedicated to Keras only.

Then we have tried to use sparkflow with tensorflow. However, we have get another problem regarding some low-level exception we couldn't fix. Actually, we have spent most of our time trying to debug those problems.

Finally, we have opted for mllib with Pyspark, which is a Python library that allows to implement many Machine Learning algorithms but doesn't allow for Neural Network models.

Anyway, we are satisfied by the mllib package for the problem we have tackled, and we have found the DVC versioning tool very interesting for a Data Scientist.