Credit Card Fraud Detection – Project Report

Data Intensive Computing – ID2221 – Group 29

Adriano Mundo (mundo@kth.se), Riccardo Colella (colella@kth.se)

October 25, 2020

1. Introduction

We live in a digital world and most of the financial transactions are executed digitally online, hence credit cards and other systems are involved. Fraudulent transactions are about 0.1% of transactions but they generate huge financial losses. Hence, banking and insurance industries have enormous interest in finding new ways to detect and anticipate these frauds. Nowadays, huge amount of data, new tools for processing big data and machine learning techniques make this possible. The aim of this project is to address the fraud detection issue by providing an architecture which is able to detect fraud in real time, as soon as they arrive.

2. Problem and Solution

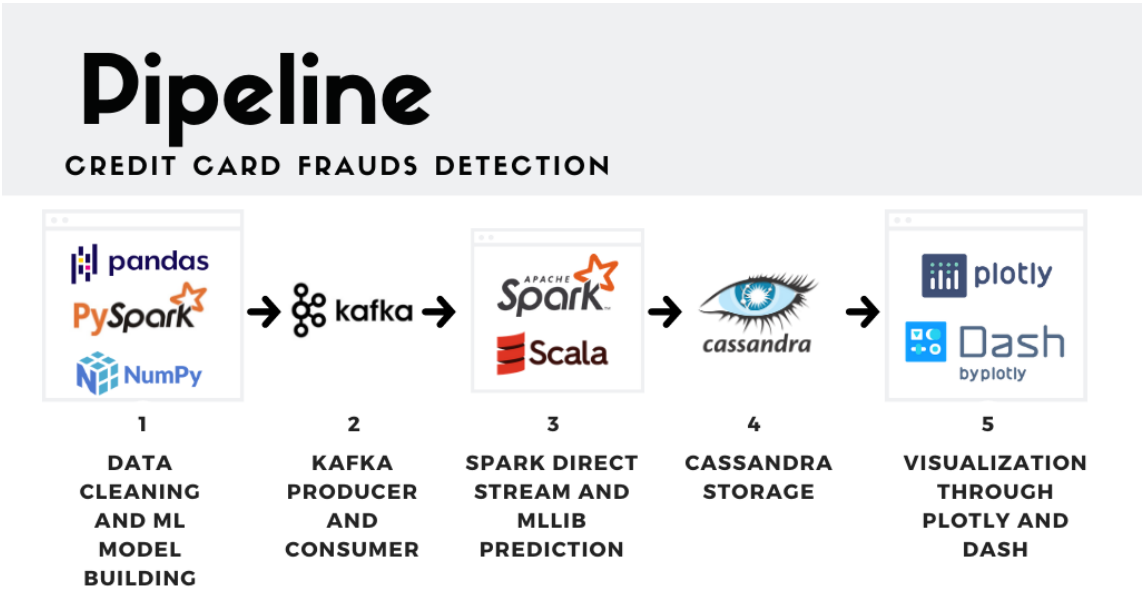
The problems that we focused on, as stated before, is detecting credit card fraud. This is a term used to point out a fraud (or a theft) committed using a payment card such as a credit or a debit card. Thus, in order to solve it we developed a system able to detect if a new transaction is fraudulent or not. Our scalable architecture can handle real time data, perform a prediction on the fly about the new transactions, evaluate, store it in a database and then visualise the final result on a dashboard.
We developed a system which generate a continuous stream of data which imitates the real data generated from credit card transactions. A machine learning has also been developed using data about past transactions and loaded in the pipeline to let the system perform the prediction on the new transactions [1]. Since this kind of data are not public, to show the real-time flows we used part of the past transactions to reproduce the streaming. Through this solution we can put into production our architecture.

3. Architecture

To achieve the proposed results, we used different framework/tools. The main ones we employed are:

- Programming languages: Python, Scala
- Data management: SparkStreaming, Kafka, Cassandra
- Machine Learning: Jupyter Notebook, PySpark, MLib
- Visualization: Dash, Plotly

As we said, our implementation comprises of a big data architecture built upon the previous mentioned tools. Data have been processed before, then a logistic regression classifier was built and stored locally. Once that, the model has been deployed on the pipeline to classify data coming from the generator. Then, they are stored in Cassandra, from where data for visualization are retrieved periodically by Dash. An image which represents our pipeline is presented below.



4. Methodology

In this section we will provide an explanation of the various steps we have done during the project, which are also the core building blocks of our architecture. It can be divided in five different parts.

4.1 Data and Machine Learning

First of all, we processed and analyzed the dataset [1]. This step is crucial in order to have a pipeline which gave meaningful result. A data exploration phase has been done to understand the data to process. Consequently, a data preparation phase was done to have all the numerical values scaled since all the features of the dataset are scaled. We made a split between data for train and test. We noticed the dataset has the problem of class imbalance; it is a highly unbalanced dataset since only 0.17% are fraudulent transactions. They are more rare than regular ones, thus we needed to create a balanced train set. The balanced dataset was used to train a Logistic Regression model to classify the transactions [3]. Refer to the Jupyter Notebook *data_exploration_modeling* in the project files to obtain more information on this phase. In the notebook all the information about the data are also provided.

4.2 Kafka communication

A part of the initial dataset, in our case the test set has been used by the Kafka [4] producer in order to mock real time transactions.
The job of the producer was to read transactions from the data previously provided and send them to Kafka on a specific topic *credit-transactions*. So, the input stream has been simulated reading from file. All the transactions sent to the topic are processed by a SparkStreaming [2] consumer which reads and puts them in a Spark Discretized Stream every batch time.

4.3 Data Stream and Prediction

This step was crucial to handle new incoming transactions. Transactions are received into a Discretized Stream by using SparkStreaming [2] and then they are transformed into DataFrames. On this DataFrames have been made some transformations to prepare the input for the model. The Logistic Regression model previously obtained is loaded into the pipeline and the transformed DataFrames are used as input for it. Therefore, for each transaction it is calculated the estimated fraudulency.
After that, transactions are cleaned, hence useless informations are dropped and it is kept only a numerical identifier of the transaction, its label (whether the transaction was fraudulent or not) and the algorithm's prediction.

4.4 Data Storage

A connection to Cassandra [5] is handled by the SparkStreaming [2] consumer and the NoSQL database is prepared to host all the transactions processed. A specific namespace *credit* is created and a table *transaction* is created too. The transaction's information, which are id, label and prediction are saved into Cassandra, hence they are available for retrieval by external applications, in our case it will be a visualization.

4.5 Visualization

While the data are stored in our database, they are continuously retrieved by our dashboard to display the result of the pipeline in real time, following a pre-defined interval refresh.
To create a dashboard, we used Dash [6], which let us create a web dashboard suited to host python code from the Plotly graphic library [7], an open source library for data visualization. For the sake of simplicity, the application is hosted locally and not deployed on a web server.
The dashboard shows the transactions saved in the database until that moment through three different view: the first shows the number of transactions predicted as fraudulent and not, the second one shows the number transactions that actually were fraudulent and not, while the third shows the percentage of correct and wrong predictions. Thanks to the dashboard built, while the whole system runs, it is possible to see the processed results changing over time.

5. How to Run

In this section it is explained how to run the code provided with this report. The recommended environment is Ubuntu >= 18.04 LTS. It is also requested to have installed on the machine Java 8 SDK, Python3 and Scala as pre-requisites.

5.1 Jupyter Notebook

To effectively run the provided jupyter notebook, it is required to have installed it on the machine. Once it has been done, a module to help location Spark in your environment from the notebook can be used and installed with the following command.

```
$ pip install findspark
```

After that, simply run the following command and navigate through the folder with the notebook named *data_ exploration_ modeling.ipynb*

```
$ jupyter notebook
```

5.2 Kafka

To run the project Kafka has to be installed in the system and the following environmental variables should be set.

```
export KAFKA_HOME="/path/to/kafka/folder"

export PATH=$KAFKA_HOME/bin:$PATH
```

After that, ZooKeeper and Kafka server have to be started

```
$KAFKA_HOME/bin/zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties

$KAFKA_HOME/bin/kafka-server-start.sh $KAFKA_HOME/config/server.properties
```

5.3 Cassandra

Cassandra has to be installed in the environment. Remember to also have installed Python2 to run the Cassandra shell. The following environmental variables have to be set

```
export CASSANDRA_HOME="/path/to/cassandra/folder"

export PYTHONPATH="/path/to/python/path"

export PATH=$PYTHONPATH_HOME/bin:$CASSANDRA_HOME/bin:$PATH
```

Before running the program, Cassandra has to be started in the foregrounds.

```
$CASSANDRA_HOME/bin/cassandra -f
```

5.4 Producer and Consumer

To run producer and consumer written in Scala, an installation of *sbt* is required.

```
echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a
/etc/apt/sources.list.d/sbt.list sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
--recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
sudo apt-get update sudo apt-get install sbt
```

For running the producer and consumer files is needed to execute the following command in the *Consumer* and *Producer* folder, assuring that each folder contains the proper *build.sbt* file.

```
$ sbt run
```

5.5 Dashboard

To run the dashboard correctly, Python3 is needed and some dependencies could be downloaded through the following commands

```
$ pip install cassandra-driver
$ pip install plotly==4.12.0
$ pip install "notebook>=5.3" "ipywidgets>=7.2"
$ pip install dash==1.16.3
```

The, it is possible to run the dashboard by executing the command from the folder where it is located the file *viz.py*.

```
$ python3 viz.py
```

6. Conclusion

In this project we created a system of real-time fraud detection. We learnt that this kind of systems are really efficient and should be used to increase the reliability of providers in real-life scenario. The big-data pipeline and the logistic regression in credit cards activities classification within the appropriate dashboard make easy to monitor the results obtained until that moment. For full code, please refer to the notebook and the files provided in the folder.

References

[1] ULB Transactions Data, https://www.kaggle.com/mlg-ulb/creditcardfraud

[2] Spark Streaming Documentation, , https://spark.apache.org/docs/latest/streaming-programming-guide.html

[3] SparkML Guide, https://spark.apache.org/docs/1.2.2/ml-guide.html

[4] Kafka Documentation, https://kafka.apache.org/documentation/

[5] Cassandra Documentation, https://cassandra.apache.org/doc/latest/

[6] Dash User Guide, https://dash.plotly.com

[7] Plotly Python Library, https://plotly.com/python/