ID2221 Project Report: Distributed Fraud Detection using Spark ML

Emma Lind and Mattias Stahre

October 2020

1 Problem description

Fraud detection is a set of activities undertaken to prevent for instance, money from being obtained through false pretenses. Fraud detection can be applied in banking, fraud may include forging checks or using stolen credit cards. The focus of this project is credit card fraud detection. The goal with this project is to classify transactions using machine learning models, the important point is that we will analyse big data and need to distribute the processing on different nodes. To handle this problem we will use Spark ML. In other words, we will classify credit card transactions on a distributed system using machine learning and spark processing. This is a typical classification problem, hence we will test two different classification machine learning algorithms. More precisely, Decision Tree and Random Forest and compare their accuracy.

2 Dataset

The data set used in this project is from Kaggle [1]. The data is labeled and in a csv format with each row correspond to a transaction and each column correspond to a feature of the transaction, for instance, time. Number of features used were 28.

3 Method

Our method for solving this problem was the following;

- Firstly, upload the data to hdfs.
- Then create a program `models.scala` that evaluates how many features to use and then train the models and upload them to hdfs.
- Further, create another program `main.scala` that reads the models from hdfs and then detects frauds and prints those results to the user. It also saves the results/predictions to hdfs.

- Lastly, create a program `evalResults.py` that evaluates the results by comparing the predictions with the labeled data using a couple of metrics such as false positive, false negative and error rate.

The main development tools were Scala, the Apache Spark API and hdfs for data storage [2].

4 Results

When evaluating the two models a number of techniques were used. Error rate in percentage, false positive and false negative. Additionally we wanted to print the detected frauds from the data set.

When developing the code we started by simply using all the parameters in the data set to train our model. This gave poor results so we implemented our own algorithm to test how many of the parameters to use. The result from this was better but it turned out that the models we used were not deterministic which means that the suggested number of features could vary. We decided to use an arbitrary model from the training program.

As for the results of the models the following was found.

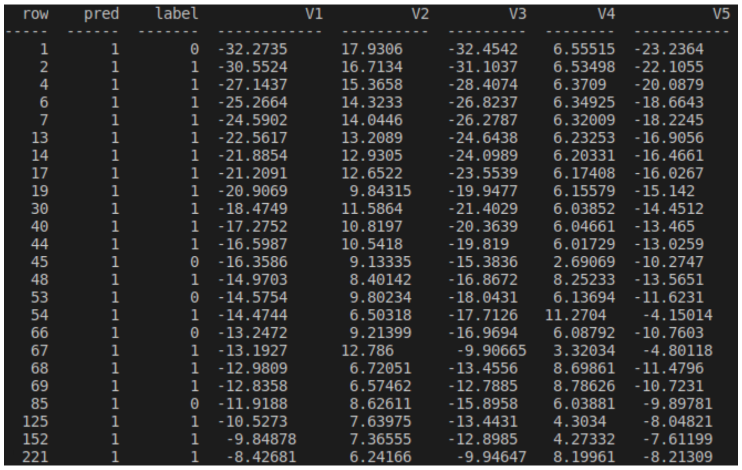
For the Decision Tree Classifier the error rate was low at 0.0747%. The false positive was also low at 0.0152%. False negative were much higher at 34% which seem to indicate that the model misses out on many transactions that were actually frauds but does not often label a non-fraudulent transaction as fraud.

As for Random Forest Classifier the error rate was low at 0.0666%. The false positives were also low at 0.0175%. Looking at the false negatives though the percentage was even higher than for the Decision Tree Classifier at 28%. This again means that the model tends to not get it wrong it terms of when frauds are not fraudulent but it misses out, in this model, on almost half of the transactions that were actually frauds.

This comparison shows that even though the error rate in terms of the total number of transactions is low for both of the models the number of missed fraudulent transactions is quite high. The reason for this is probably because of how often non-frauds occur in the data set compared to frauds. Because most of the transactions in the data set are non-fraudulent and both models are quite good at not giving false positives, evaluating the models against all transactions is not a good way to evaluate them.

Having trained a model our program for analysis takes a csv-file, which contains transactions, and returns a list of the suspected frauds. The identifier for each transaction corresponds to the row in the input data.

The figure below show that often the models predict fraud to be true (indicated by integer 1) although the label is actually false (indicated by integer 0).



```
  row    pred    label           V1          V2         V3        V4           V5
-----  ------  -------  -----------  ----------  ---------  --------  -----------
    1       1        0  -32.2735     17.9306     -32.4542    6.55515  -23.2364
    2       1        1  -30.5524     16.7134     -31.1037    6.53498  -22.1055
    4       1        1  -27.1437     15.3658     -28.4074    6.3709   -20.0879
    6       1        1  -25.2664     14.3233     -26.8237    6.34925  -18.6643
    7       1        1  -24.5902     14.0446     -26.2787    6.32009  -18.2245
   13       1        1  -22.5617     13.2089     -24.6438    6.23253  -16.9056
   14       1        1  -21.8854     12.9305     -24.0989    6.20331  -16.4661
   17       1        1  -21.2091     12.6522     -23.5539    6.17408  -16.0267
   19       1        1  -20.9069      9.84315    -19.9477    6.15579  -15.142
   30       1        1  -18.4749     11.5864     -21.4029    6.03852  -14.4512
   40       1        1  -17.2752     10.8197     -20.3639    6.04661  -13.465
   44       1        1  -16.5987     10.5418     -19.819     6.01729  -13.0259
   45       1        0  -16.3586      9.13335    -15.3836    2.69069  -10.2747
   48       1        1  -14.9703      8.40142    -16.8672    8.25233  -13.5651
   53       1        0  -14.5754      9.80234    -18.0431    6.13694  -11.6231
   54       1        1  -14.4744      6.50318    -17.7126   11.2704     -4.15014
   66       1        0  -13.2472      9.21399    -16.9694    6.08792  -10.7603
   67       1        1  -13.1927     12.786       -9.90665   3.32034    -4.80118
   68       1        1  -12.9809      6.72051    -13.4556    8.69861  -11.4796
   69       1        1  -12.8358      6.57462    -12.7885    8.78626  -10.7231
   85       1        0  -11.9188      8.62611    -15.8958    6.03881    -9.89781
  125       1        1  -10.5273      7.63975    -13.4431    4.3034     -8.04821
  152       1        1   -9.84878     7.36555    -12.8985    4.27332    -7.61199
  221       1        1   -8.42681     6.24166     -9.94647   8.19961    -8.21309
```

Figure 1: Figure showing part of the false positives for Decision Tree Classifier.

5 How to run the code

First start the namenode and the datanode:

```
hdfs --daemon start namenode
hdfs --daemon start datanode
```

Upload the data to hdfs:

```
hdfs dfs -mkdir /kth
hdfs dfs -put creditcard.csv /kth
```

Check so it is uploaded:

```
hdfs dfs -ls /kth
```

Then run the first program `models.scala` to determine how many features to use and to train models: cd/trainModels

```
sbt run
```

Then run the main program `main.scala` to test the models: cd/id2221

```
sbt run
```

Lastly run the python program `evalResults.py` to evaluate the results:

```
python3 evalResults.py
```

References

[1] Dileep, "Credit card fraud detection," Available at https://www.kaggle.com/dileep070/anomaly-detection.

[2] Available at https://spark.apache.org/.