Real-Time Fraud Detection on Credit Cards Transactions

Prasanna Surianarayanan, Dipika Rajesh

ps3703@nyu.edu, dr2898@nyu.edu

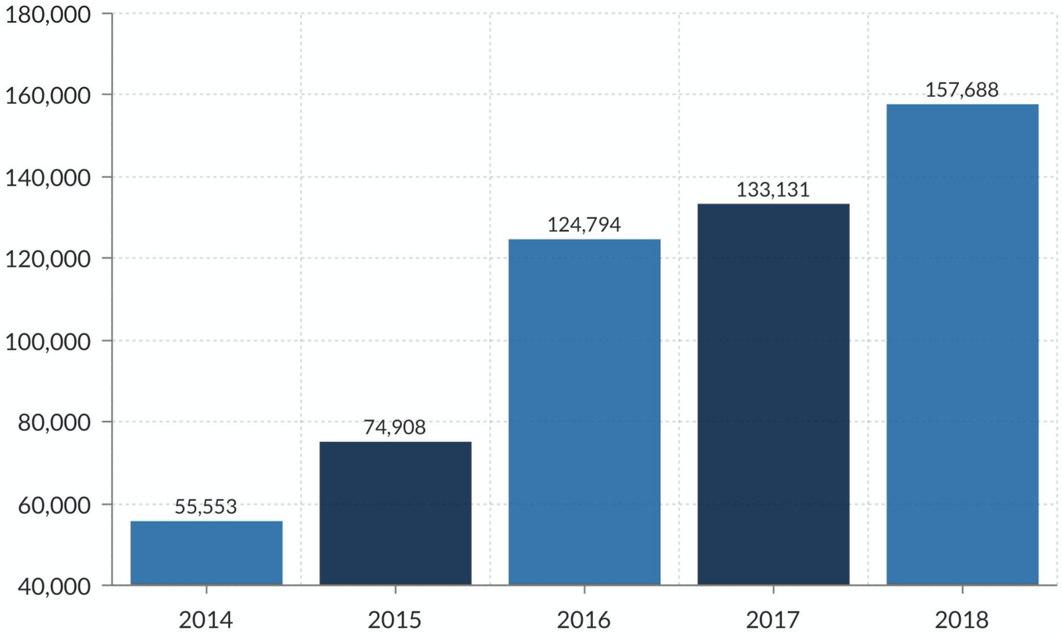
Premise

- Fraud detection applies to a variety of sectors:
  - Banking and Finance
  - Insurance
  - Government agencies and law enforcement
- 2018 PwC Survey : fraud experienced by 49% of the 7,200 companies surveyed
- Fraud Detection : A billion-dollar business

Need for Credit Card Fraud Detection

- Electronic commerce has rapidly increased over the years
- Increasing confidence of customers in electronic payments
- Customers increasingly vulnerable to new attacks
- Frauds need to be detected in (near) real time
- Scalable learning techniques needed to ingest and analyse massive amounts of streaming data.

Credit Card Fraud Reports in the United States



SHIFT
Credit Card Processing

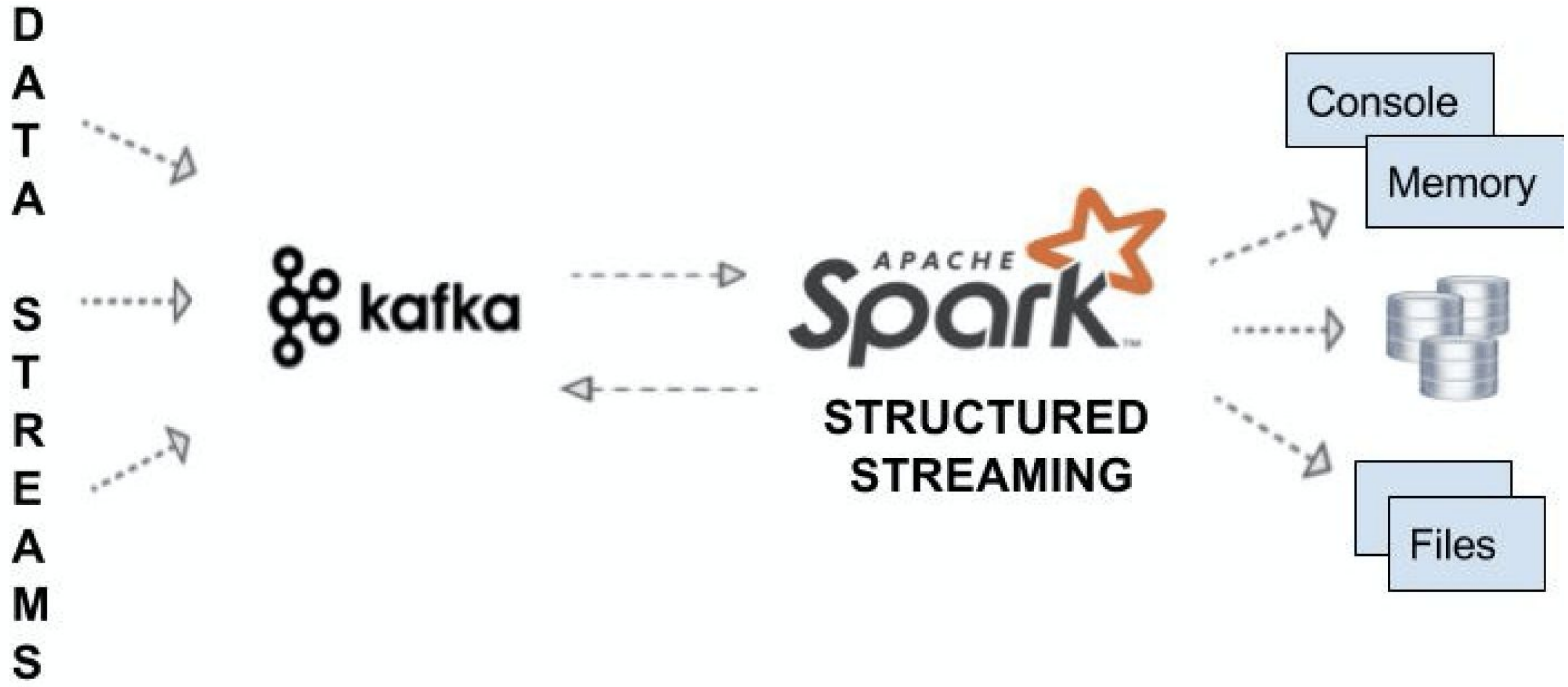
Leveraging Big Data for the Use Case

- Credit card transactions captured over time based on the cardholder’s spending behaviour
- Events stored as transaction history for detection of fraud in the future
- Spark Structured Streaming and Apache Kafka work pretty well in capturing such incidents real time
- Ecosystem capable of handling vast data with good throughput and low latency
- SparkML used to develop the prediction model

Technologies Used

- Apache Kafka for message ingestion
- Spark Structured Streaming to process datastreams from Kafka
- Spark ML to detect fraudulent transactions

Proposed Architecture



Ecosystem Overview

- **Apache Kafka** :
  - Transactions ingested to Kafka
  - Used for delivering streams of card transactions
  - Card transactions are published to a topic
  - Consumers read messages from subscribed topics

Ecosystem Overview

- **Spark Structured Streaming** :
  - Reads stream of data from the Kafka topic
  - Creates dataframes in streaming mode
  - Treats arriving data as an unbounded input table
  - New items in streams treated like rows, appended to the input table
  - Queries applied to streaming dataframes just like static dataframes
  - Triggers used to control result updates

Ecosystem Overview

- **Spark ML :**
  - Used to create Models to detect fraudulent transactions
  - Historical transactions along with the class labels are used to train the model
  - Prediction is performed on the incoming data from the Kafka topic
  - Predicted results written to an output sink

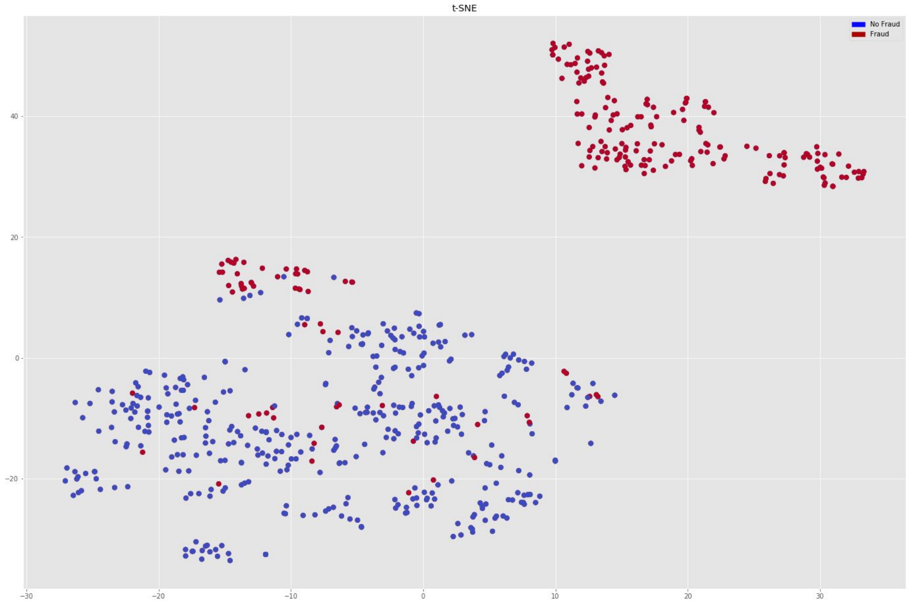
Dataset Used

- Source of labelled data : Kaggle - https://www.kaggle.com/mlg-ulb/creditcardfraud
- Total features : 31
- 28 features transformed using PCA for confidentiality purposes
- 3 Non-anonymized features : Time, Amount, Class Label
- Dataset was selected due to non-synthetic, high-dimensional data.

Data Exploration!

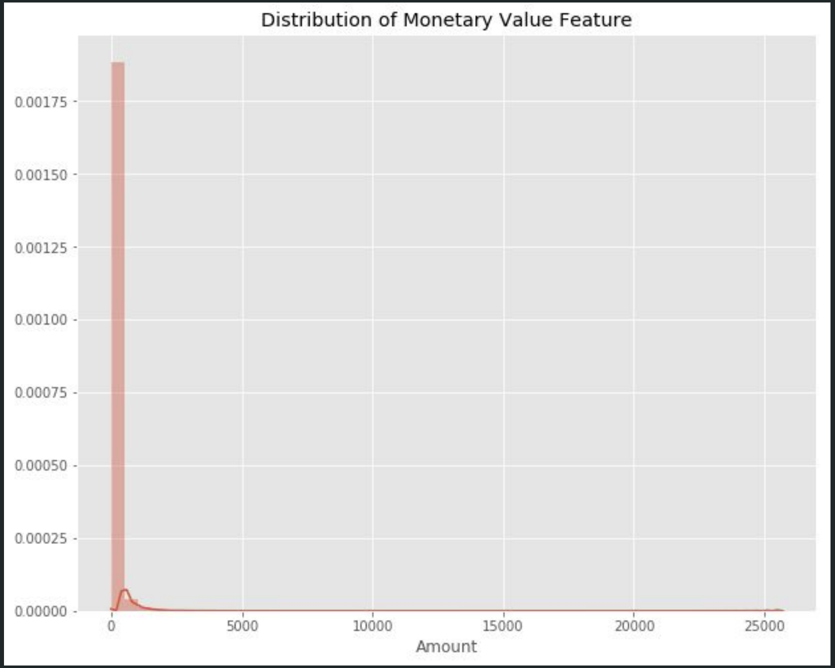
Dimensionality Reduction using t-SNE

T-distributed Stochastic Neighbor Embedding (t-SNE) nonlinear dimensionality reduction technique well-suited for embedding high-dimensional data for visualization in a low-dimensional space of two or three dimensions

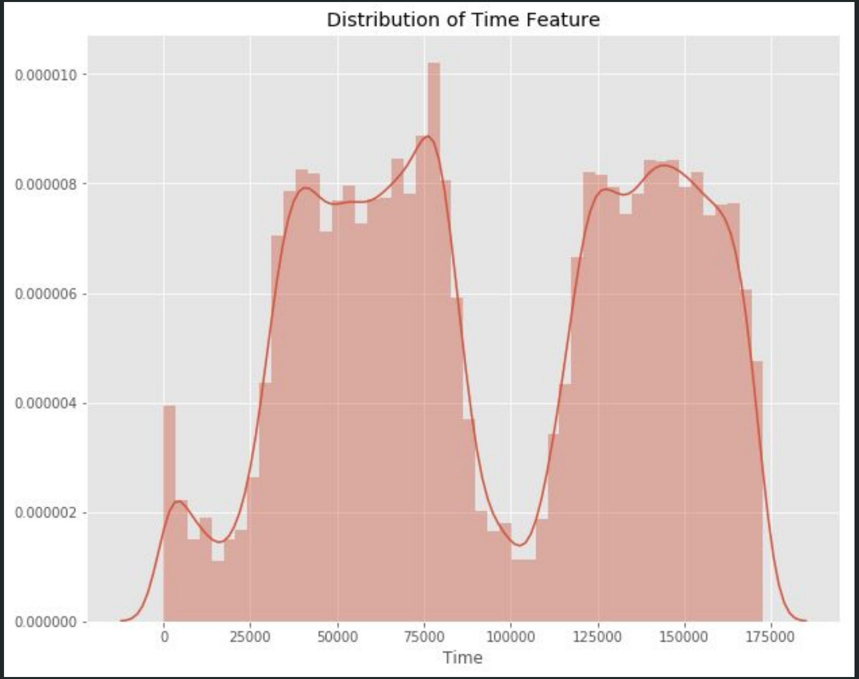


Exploration of the Dataset

The vast majority of transactions are relatively small and only a tiny fraction of transactions comes even close to the maximum - needs to be scaled!

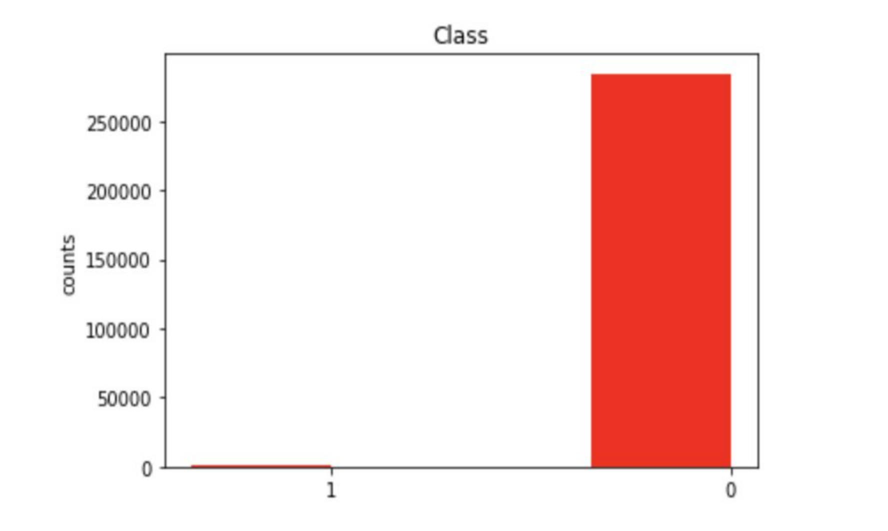


- The time is recorded in the number of seconds since the first transaction in the data set.
- Can conclude that this data set includes all transactions recorded over the course of two days.



Class Distribution?

Most transactions are non-fraudulent. In fact, 99.83% of the transactions in this data set were not fraudulent while only 0.17% were fraudulent.

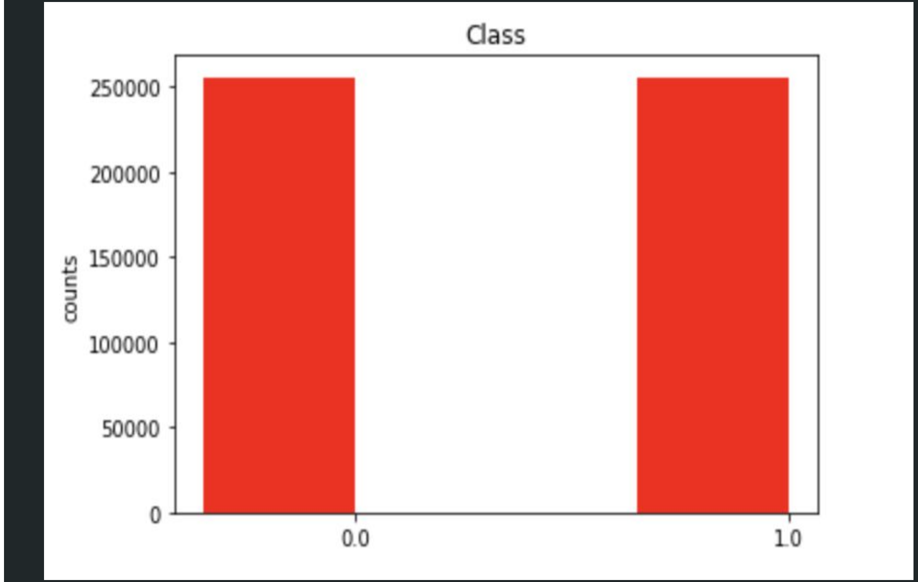


Data Preprocessing!

- Imbalanced datasets can create biased machine learning models
- Oversampling methods (SMOTE) were applied on the dataset along with feature scaling.

After applying SMOTE,

the oversampling created balanced classes!

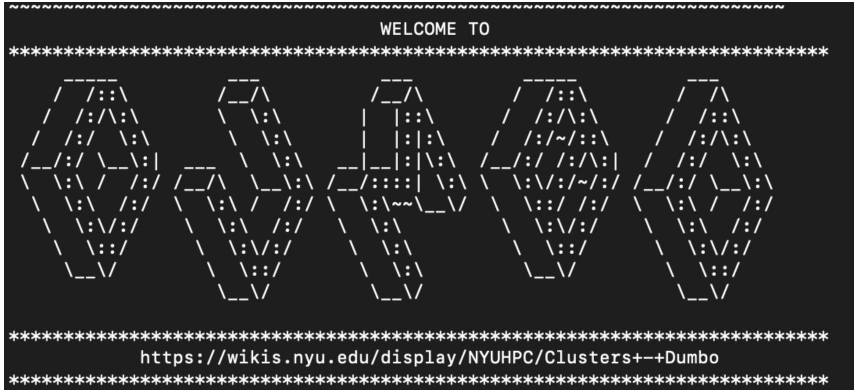


Choosing and Fitting the Machine Learning model

Model Choice - Random Forest Classifier

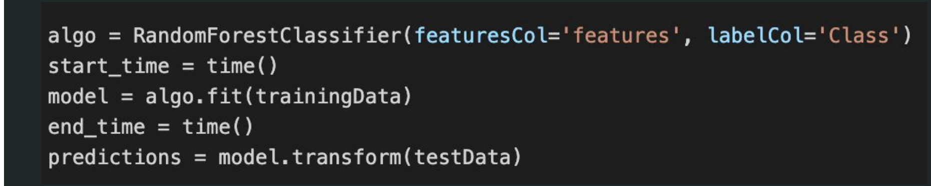
Advantages of Random Forest Classifiers:

- Generalizes patterns well
- Robust to different input types
- Robust to outliers
- Highly scalable



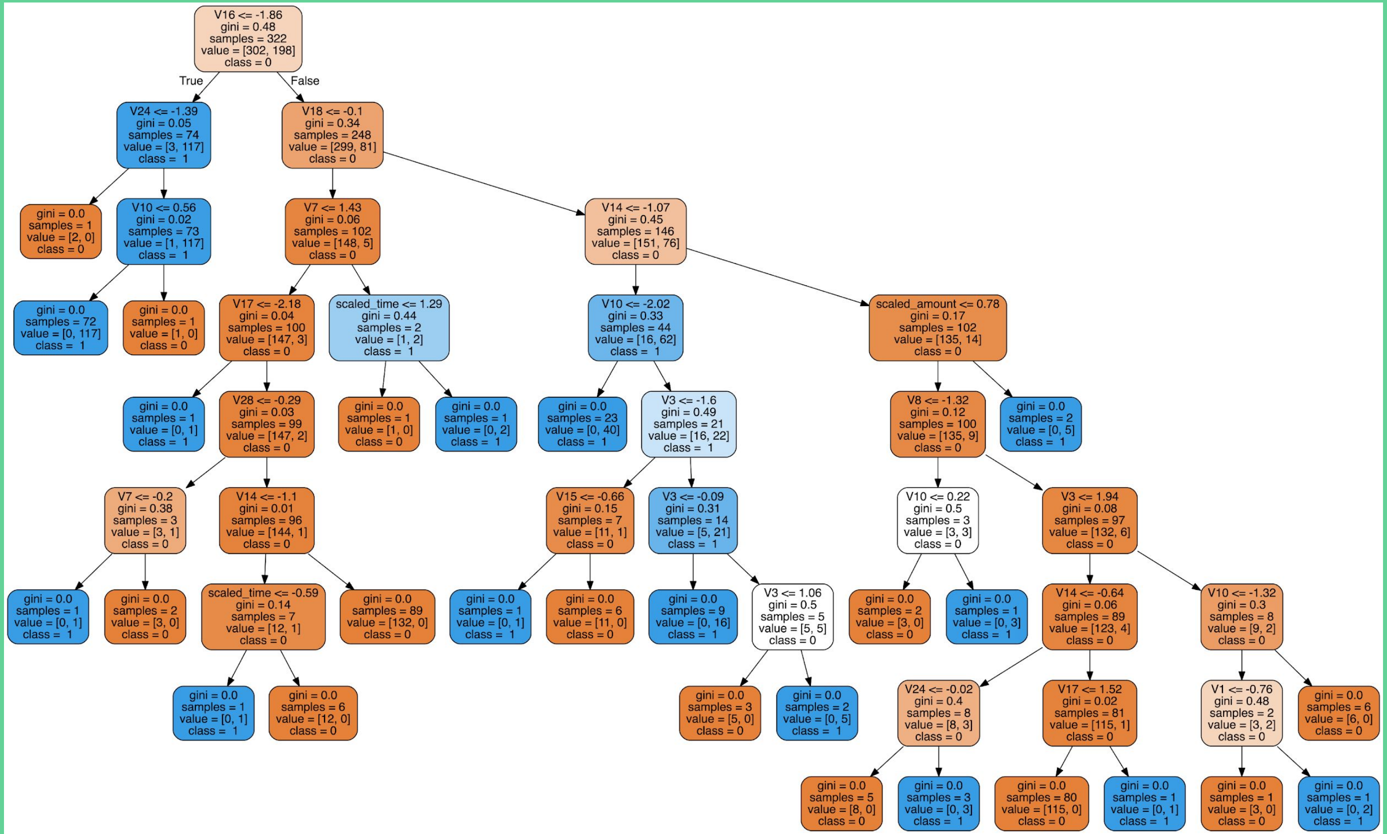
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                  WELCOME TO
****************************************************************************

     _____          ___           ___           _____          ___
    /  /::\        /__/\         /__/\         /  /::\        /  /\
   /  /:/\:\       \  \:\       |  |::\       /  /:/\:\      /  /::\
  /  /:/  \:\       \  \:\      |  |:|:\     /  /:/~/::\    /  /:/\:\
 /__/:/ \__\:|  ___  \  \:\   __|__|:|\:\   /__/:/ /:/\:| /  /:/  \:\
 \  \:\ /  /:/ /__/\  \__\:\ /__/::::| \:\  \  \:\/:/~/:/ /__/:/ \__\:\
  \  \:\  /:/  \  \:\ /  /:/ \  \:\~~\__\/   \  \::/ /:/  \  \:\ /  /:/
   \  \:\/:/    \  \:\  /:/   \  \:\          \  \:\/:/    \  \:\  /:/
    \  \::/      \  \:\/:/     \  \:\          \  \::/      \  \:\/:/
     \__\/        \  \::/       \  \:\          \__\/        \  \::/
                   \__\/         \__\/                        \__\/

****************************************************************************
            https://wikis.nyu.edu/display/NYUHPC/Clusters+-+Dumbo
****************************************************************************
```

Using RandomForestClassifier from SparkML library, trained on NYU DUMBO HPC with 500,000 data instances!

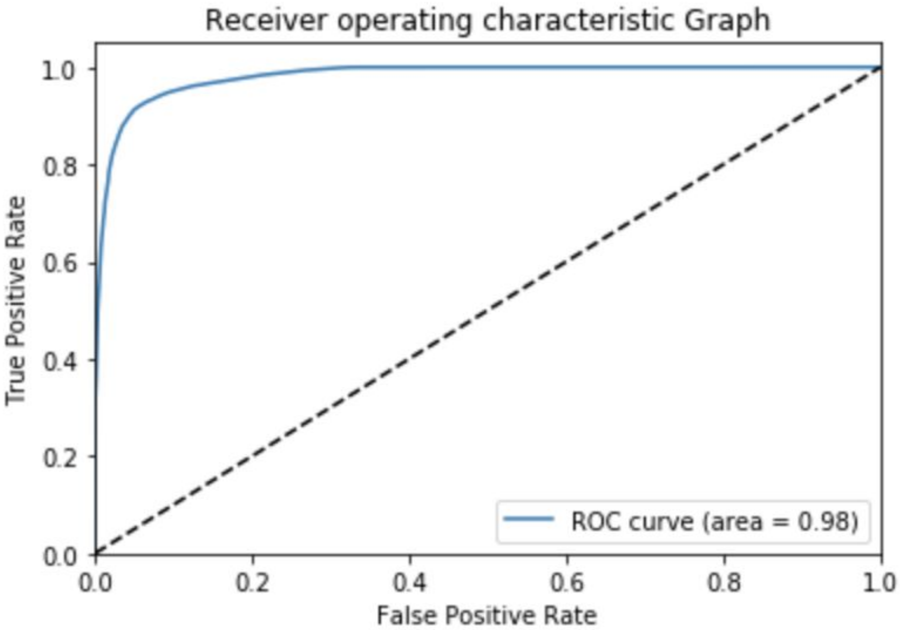


```
algo = RandomForestClassifier(featuresCol='features', labelCol='Class')
start_time = time()
model = algo.fit(trainingData)
end_time = time()
predictions = model.transform(testData)
```

Visualizing the Random Forest model using Eureka Trees library

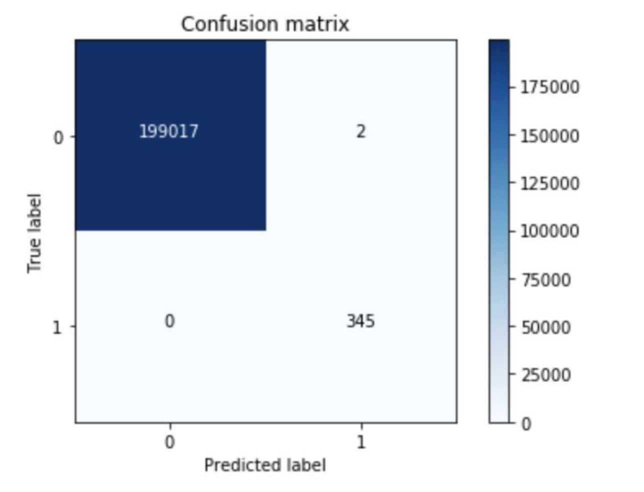

V16 <= -1.86 gini = 0.48 samples = 322 value = [302, 198] class = 0
True
False
V24 <= -1.39 gini = 0.05 samples = 74 value = [3, 117] class = 1
V18 <= -0.1 gini = 0.34 samples = 248 value = [299, 81] class = 0
gini = 0.0 samples = 1 value = [2, 0] class = 0
V10 <= 0.56 gini = 0.02 samples = 73 value = [1, 117] class = 1
V7 <= 1.43 gini = 0.06 samples = 102 value = [148, 5] class = 0
V14 <= -1.07 gini = 0.45 samples = 146 value = [151, 76] class = 0
gini = 0.0 samples = 72 value = [0, 117] class = 1
gini = 0.0 samples = 1 value = [1, 0] class = 0
V17 <= -2.18 gini = 0.04 samples = 100 value = [147, 3] class = 0
scaled_time <= 1.29 gini = 0.44 samples = 2 value = [1, 2] class = 1
V10 <= -2.02 gini = 0.33 samples = 44 value = [16, 62] class = 1
scaled_amount <= 0.78 gini = 0.17 samples = 102 value = [135, 14] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
V28 <= -0.29 gini = 0.03 samples = 99 value = [147, 2] class = 0
gini = 0.0 samples = 1 value = [1, 0] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1
gini = 0.0 samples = 23 value = [0, 40] class = 1
V3 <= -1.6 gini = 0.49 samples = 21 value = [16, 22] class = 1
V8 <= -1.32 gini = 0.12 samples = 100 value = [135, 9] class = 0
gini = 0.0 samples = 2 value = [0, 5] class = 1
V7 <= -0.2 gini = 0.38 samples = 3 value = [3, 1] class = 0
V14 <= -1.1 gini = 0.01 samples = 96 value = [144, 1] class = 0
V15 <= -0.66 gini = 0.15 samples = 7 value = [11, 1] class = 0
V3 <= -0.09 gini = 0.31 samples = 14 value = [5, 21] class = 1
V10 <= 0.22 gini = 0.5 samples = 3 value = [3, 3] class = 0
V3 <= 1.94 gini = 0.08 samples = 97 value = [132, 6] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 2 value = [3, 0] class = 0
scaled_time <= -0.59 gini = 0.14 samples = 7 value = [12, 1] class = 0
gini = 0.0 samples = 89 value = [132, 0] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 6 value = [11, 0] class = 0
gini = 0.0 samples = 9 value = [0, 16] class = 1
V3 <= 1.06 gini = 0.5 samples = 5 value = [5, 5] class = 0
gini = 0.0 samples = 2 value = [3, 0] class = 0
gini = 0.0 samples = 1 value = [0, 3] class = 1
V14 <= -0.64 gini = 0.06 samples = 89 value = [123, 4] class = 0
V10 <= -1.32 gini = 0.3 samples = 8 value = [9, 2] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 6 value = [12, 0] class = 0
gini = 0.0 samples = 3 value = [5, 0] class = 0
gini = 0.0 samples = 2 value = [0, 5] class = 1
V24 <= -0.02 gini = 0.4 samples = 8 value = [8, 3] class = 0
V17 <= 1.52 gini = 0.02 samples = 81 value = [115, 1] class = 0
V1 <= -0.76 gini = 0.48 samples = 2 value = [3, 2] class = 0
gini = 0.0 samples = 6 value = [6, 0] class = 0
gini = 0.0 samples = 5 value = [8, 0] class = 0
gini = 0.0 samples = 3 value = [0, 3] class = 1
gini = 0.0 samples = 80 value = [115, 0] class = 0
gini = 0.0 samples = 1 value = [0, 1] class = 1
gini = 0.0 samples = 1 value = [3, 0] class = 0
gini = 0.0 samples = 1 value = [0, 2] class = 1

Results! - Area Under the Receiver-Operating Characteristic (AUROC) metric

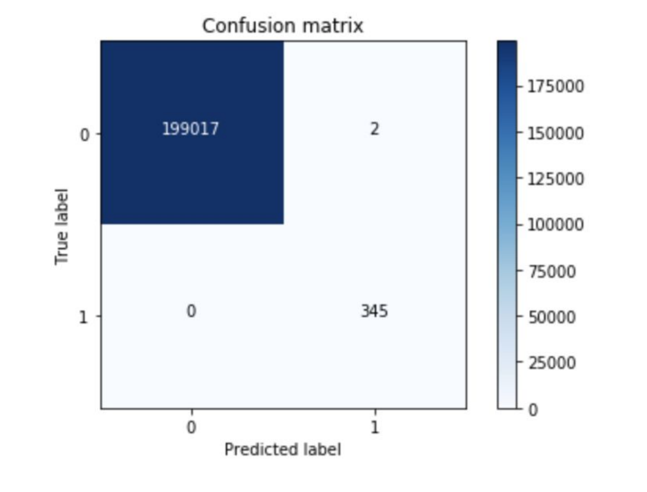


AUC — ROC Curve

Evaluation of the random forest model - confusion matrix



Recall in the train set = 1.0



Recall in the test set = 0.8095

Never use accuracy score as an evaluation metric on imbalanced datasets - will be high and misleading.
Never test the model on the sampled (over or under) data.

Future Work

- Adding ElasticSearch and Kibana to the pipeline for visualization and trend analysis
- Expanding model to include Time-Series Analysis of Customer Credit Card Transactions
- Online Machine Learning with incoming real-time data

Thank you!