



# Data Streaming and Credit Card Fraud Detection model

# Agenda

- Introduction
- EDA (Exploratory Data Analysis) and Data Preprocessing
- Implementation Framework
- Scale comparison & evaluation
- Performance Measurements
- Conclusion



# Introduction

## EMR Analytics

- Train 4 models on different algorithms
- Make predictions and evaluate accuracy

## Local v.s Distributed

Compare training and test time



**PySpark**

Use PySpark on AWS to separate data to train and test dataset

**Data Streaming**

Every 30 seconds streaming 5,000 data rows and make predictions

## Future

- Data Staging
- Store data to MongoDB
- Kafka
- Virtualization by Tableau/Power BI

# Dataset Overview

- From Kaggle. The size of the data:

```
In [6]: data.shape
```

```
Out[6]: (284807, 31)
```

- Missing data:

```
In [13]: data.isnull().sum().any()
```

```
Out[13]: False
```

- Check how imbalanced the dataset is

```
+-----+-----+
| Class | count |
+-----+-----+
|      0 | 227429 |
|      1 |    417 |
+-----+-----+
```

- The brief summary of the dataset:

```
df_sub1.describe().show()
```

```
+-----+-----+-----+-----+
| summary | Amount | Class | Time |
+-----+-----+-----+-----+
| count | 227846 | 227846 | 227846 |
| mean | 90.82474026317597 | 0.001830183545026... | 79043.08588695874 |
| stddev | 250.50323615893055 | 0.04274157216455668 | 39506.09954343867 |
| min | 0.0 | 0 | 0.0 |
| max | 19656.53 | 1 | 145248.0 |
+-----+-----+-----+-----+
```

# Resampling

- Implement both **SMOTE** (Synthetic Minority Over-sampling Technique) and **Down Sampling** in PySpark to re-sample the dataset.
- 10k** fraud v.s. **10k** normal
- 80%** training set & **20%** test set. We also retain **10%** of the data for simulating the streaming process.

## 2. Down-sampling

```
In [8]: # down sampling to meet the SMOTE
df_down = df.filter("Class=0").rdd.takeSample(False,10332, seed=0)

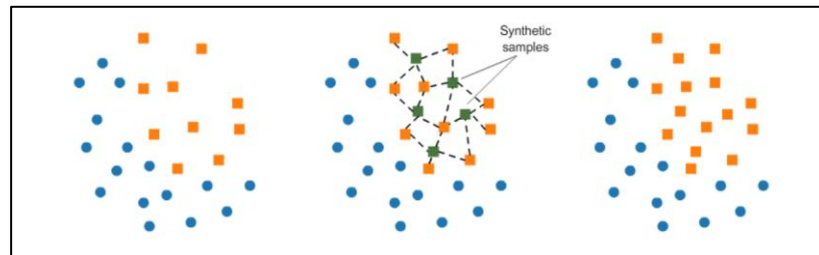
In [10]: # convert the list to dataframe for outfile
df_down1 = sqlContext.createDataFrame(df_down)

In [11]: df_down1.coalesce(1).write.csv('s3://storechen/678_down.csv')

In [12]: print((df_down1.count(), len(df_down1.columns)))

(10332, 31)
```

## 1. Over-sampling: SMOTE



Source: <https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>

```
def SmoteSampling(vectorized, k = 5, minorityClass = 1, majorityClass = 0, percentageOver = 200, percentageUnder = 100):
    if (percentageUnder > 100 | percentageUnder < 10):
        raise ValueError("Percentage Under must be in range 10 - 100");
    if (percentageOver < 100):
        raise ValueError("Percentage Over must be in at least 100");
    dataInput_min = vectorized[vectorized['label'] == minorityClass]
    dataInput_maj = vectorized[vectorized['label'] == majorityClass]
    feature = dataInput_min.select('features')
    feature = feature.rdd
    feature = feature.map(lambda x: x[0])
    feature = feature.collect()
    feature = np.asarray(feature)
    nbrs = neighbors.NearestNeighbors(n_neighbors=k, algorithm='auto').fit(feature)
    neighbours = nbrs.kneighbors(feature)
```

Code for implementing SMOTE in PySpark

# EDA (Exploratory Data Analysis)

- The Max & Min transaction amount

```
from pyspark.sql.functions import max,min
df_sub1.select(max("Amount"),min("Amount")).show()
```

```
+-----+-----+
|max(Amount)|min(Amount)|
+-----+-----+
| 19656.53|          0.0|
+-----+-----+
```

- Correlation

```
from pyspark.sql.functions import corr
df_sub1.select(corr("Class","Amount")).show()
print("The correlation is pretty low")
```

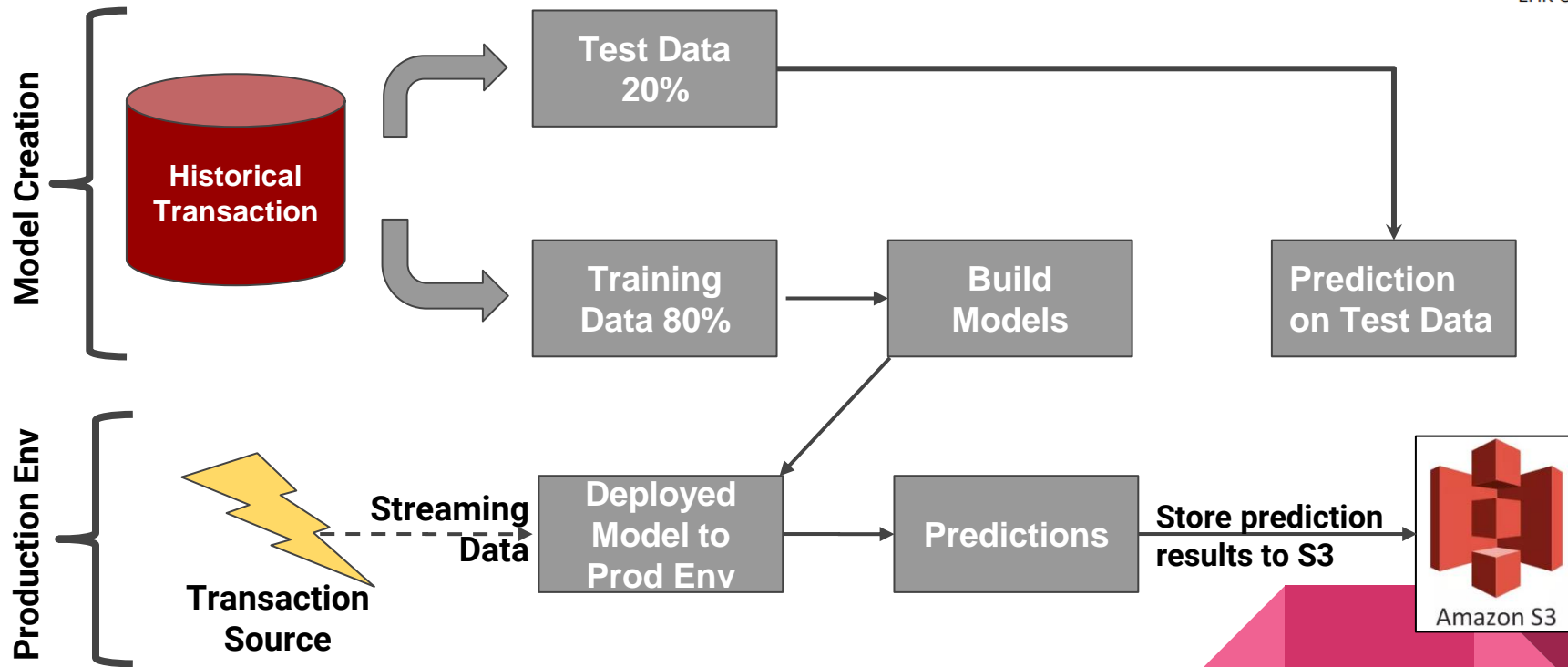
```
+-----+
| corr(Class, Amount)|
+-----+
|0.005953969765854438|
+-----+
```

- The Top 10 Transactions:

```
df_sub1.orderBy(df_sub1["Amount"].desc()).show(10)
```

```
+-----+-----+-----+
| Amount|Class| Time|
+-----+-----+-----+
|19656.53| 0| 48401.0|
| 18910.0| 0| 95286.0|
|12910.93| 0| 42951.0|
|11898.09| 0| 46253.0|
|11789.84| 0|119713.0|
| 8790.26| 0| 55709.0|
| 8360.0| 0|144755.0|
| 7879.42| 0| 30537.0|
| 7766.6| 0|128027.0|
| 7712.43| 0| 1264.0|
+-----+-----+-----+
```

# Implementation Framework








# Streaming Data in Production Environment

- Data is streamed from our local machine to S3, using a AWS SDK Boto3
- Size and the frequency of the stream data can be fully configured.
- We stream 5k transactions every 30 seconds
- After data are streamed to the production environment, system makes prediction results.

```
def upload_to_aws(local_file, bucket, s3_file):  
    s3 = boto3.client('s3', aws_access_key_id=ACCESS_KEY,  
                      aws_secret_access_key=SECRET_KEY)
```



<input type="checkbox"/>	 stream0.csv	Nov 28, 2019 7:45:14 AM GMT+0700
<input type="checkbox"/>	 stream1.csv	Nov 28, 2019 7:45:46 AM GMT+0700
<input type="checkbox"/>	 stream2.csv	Nov 28, 2019 7:46:18 AM GMT+0700
<input type="checkbox"/>	 stream3.csv	Nov 28, 2019 7:46:49 AM GMT+0700
<input type="checkbox"/>	 stream4.csv	Nov 28, 2019 7:47:21 AM GMT+0700



# Streaming Data in Production Environment

- Stream of data keeps coming into the environment where the classification model is deployed
- The program is able to produce prediction result continuously
- The prediction results are stored back to another directory on S3 for further analysis purpose such as BI tools/ Visualization etc.

```
Test result of stream0 is
Accuracy of LogisticRegression is = 0.9978
F1 of LogisticRegression = 0.998126
```

```
Test result of stream1 is
Accuracy of LogisticRegression is = 0.9996
F1 of LogisticRegression = 0.999578
```

```
Test result of stream2 is
Accuracy of LogisticRegression is = 0.9992
F1 of LogisticRegression = 0.9992
```

```
Test result of stream3 is
Accuracy of LogisticRegression is = 0.999
F1 of LogisticRegression = 0.999029
```

Search: Type a prefix and press Enter to search. Press ESC to clear.

Upload Create folder Download Actions

US East (N. Virginia)

Viewing 1 to 9

Name	Last modified	Size	Storage class
prediction0.csv	--	--	--
prediction1.csv	--	--	--
prediction2.csv	--	--	--
prediction3.csv	--	--	--
prediction4.csv	--	--	--

```
lrprediction_stream[2].show(5)
```

features	label	rawPrediction	probability	prediction
[67478.0, 1.007833...]	0	[8.67523896400722...]	[0.99982926701474...]	0.0
[67478.0, 1.267602...]	0	[6.93116946998853...]	[0.99902409552563...]	0.0
[67480.0, -1.07630...]	0	[7.98574887150973...]	[0.99965983817278...]	0.0
[67482.0, -0.79236...]	0	[6.37341728977651...]	[0.99829658829535...]	0.0
[67483.0, -1.22259...]	0	[8.22202776017821...]	[0.99973140244960...]	0.0

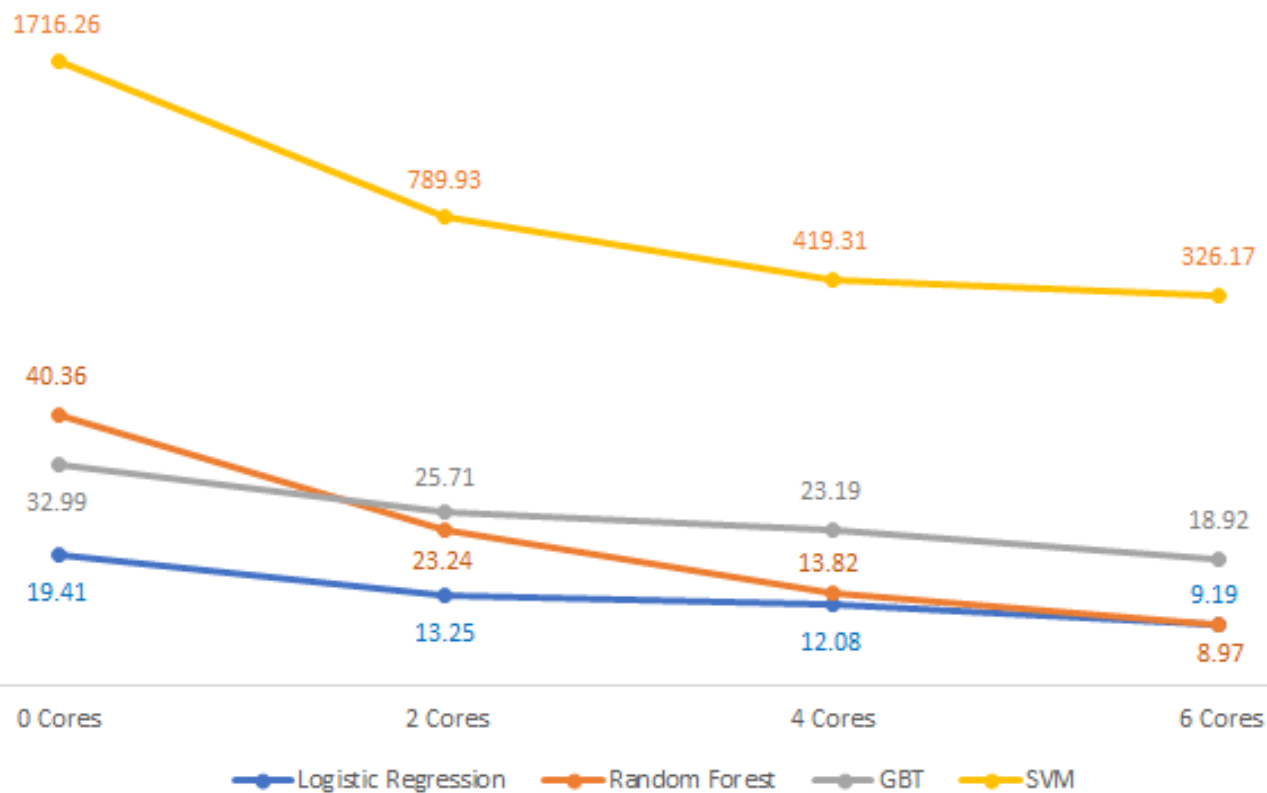
only showing top 5 rows

# Scale comparison & Evaluation

- Training data set consists of > 235K transactions, 31 features
- Measuring the training time under LogisticRegression, RandomForest, Gradient-boosted Tree, Support Vector Machine, under different number of cores.
- Parameter setting of models in each hardware sizing is identical

Time/ Cores	Logistic Regression	Random Forest	GBT	SVM
0 Core (Master only)	19.41 seconds	40.36 seconds	32.99 seconds	1716.26 seconds
2 Cores	13.25 seconds	23.24 seconds	25.71 seconds	789.93 seconds
4 Cores	12.08 seconds	13.82 seconds	23.19 seconds	419.31 seconds
6 Cores	9.19 seconds	8.97 seconds	18.92 seconds	326.17 seconds

Time (Seconds) / Cores



# Performance Measurements

- Based on the test data set consisting of ~ 4k transactions

Measuring Metrics	Logistic Regression	Random Forest	Gradient-boosted Tree	SVM
Accuracy	0.9819	0.9789	0.9816	0.9829
F1	0.9809	0.9789	0.9824	0.9819

# Conclusion & Suggestions for Future work

- Cloud computing does take less time than local computing, the training time significantly decrease with stronger hardware
- The time required to run the program is unstable
- Connection to NoSQL DB such as Cassandra or MongoDB for further processing such as Dashboard/ Report
- Full integration with Kafka for a more robust data streaming capability

