Real Time Fraud Detection With Sequence Mining on Big Data Platform

Pranab Ghosh
Big Data Consultant
IEEE CNSV meeting, May 6 2014
Santa Clara, CA
Open Source Big Data Eco System

- Query (NOSQL) : Cassandra, HBase, MongoDB and more
- Query (SQL) : Hive, Stinger, Impala, Presto, Shark
- Aggregation Analytic (NOSQL) : Druid, MongoDB
- Aggregation Analytic (SQL) : Hive, Stinger, Impala, Presto, Shark
- Advanced Analytic : Hadoop, Spark
- Real time : Storm, Samza, S4, Spark Streaming
Hadoop

- Power of functional programming and parallel processing join hands in *Hadoop*
- Parallel processing framework running on cluster of commodity machines
- Stateless functional programming because processing of each row of data does not depend upon any other row or any state
- Divide and conquer. Data gets partitioned and partitions get processed in parallel
Storm

- Clustered framework for scalable real time stream processing
- Like Hadoop, parallel processing framework runs on cluster of commodity machines
- Uses a combination of processes and threads for parallelism, unlike Hadoop which uses only processes
- Unlike 2 processing stages in Hadoop (map and reduce) there can be multiple processing stages defined in a Storm topology.
Redis

- It's a wonderful glue for Big Data eco system
- Many people think of it as a distributed data structure server
- Can be used for list, queue, cache etc.
- Supports master slave replication
- There is no sharding support
Fraud Detection Basics

- Belongs to the general category of problems known as Outlier Detection i.e., detecting data points that don’t follow the trends and patterns in the data.

- Two approaches for treating input: 1. focus on instance of data point  2. focus on sequence of data points.

- Two kinds of algorithms : 1. building a model out of data 2. using data directly.

- Real-time fraud detection is only feasible with model-based approach. A model is built with batch processing of training data. A real-time stream processor uses the model and makes predictions in real time.
Credit Card Fraud Detection

- We will use sequence based approach. We will use *Hadoop* to build a Markov Chain model.

- *Storm* will process transaction data in real time and will use the Markov Chain model to predict potential fraud in the incoming transaction stream.

- *Redis* is used for the incoming transaction queue. It’s also used as key value store for the Markov model. Fraudulent transaction sequences are written to another *Redis* queue.
Why Sequence Based Algorithm

- The sequence based algorithms are generally powerful. Certain fraudulent activities may not be detectable with instance based algorithms.
- If your bank account is hacked and there are many transactions involving withdrawal of small amount of money, instance based algorithms will fail to detect the fraud.
- However a sequence based algorithm is more likely to detect the fraudulent activities.
Architecture

1. Hadoop processing of historical transactions
2. Write markov chain model to Redis cache
3. Load markov chain model from Redis cache
4. Ingest real-time transactions
5. Write fraudulent transaction sequence
   - Storm topology
   - Storm
   - Redis
   - Cache
   - Input Queue
   - Output Queue
Building Markov Chain Model with Hadoop

- A transaction consists of the triplet (amount spent, whether it includes high price ticket item, time elapsed since the last transaction)
- Each item in the triplet is categorical and has possible 2 or 3 values. We end up with 18 possible transaction types.
- Our goal is to build a 18 x 18 state transition probability matrix
- Input data consists of customer ID, transaction ID and transaction type.
Building Markov Chain Model with Hadoop

- The data is processed through a Map Reduce job to group by customer ID, so that we can have the transaction sequence for each customer.

- The next Map Reduce job counts the different state transitions and builds the 18 x 18 state transition probability matrix.

- We are using the first order Markov model i.e., the probability of a state only depends on the earlier state.
Real Time Prediction with Storm

- Our Storm topology (i.e. job) is very simple, with one spout for ingesting transaction stream from a Redis queue. There is one bolt that does all the processing.

- The incoming transaction stream is field grouped (i.e., partitioned) by the customer ID, so that each bolt instance processes data for a subset of customers.

- Each bolt maintains a window of preconfigured size for each user, where it collects the recent n transactions.

- Every time the window gets updated, the bolt calculates certain metric indicative of whether the current transaction sequence is fraudulent.
Real Time Prediction with Storm

- There are various outlier metrics for sequence. The one we are using is called *Miss Probability Metric*.

- For a transaction state pair in the sequence, it sums all the state transition probabilities except when the target state is the second state of the pair. The process is repeated for all pair in the sequence and all probabilities are summed.

- If the final sum exceeds some predefined threshold, the transaction sequence is deemed fraudulent. The transaction sequence is written to a *Redis* queue.
Sequence Outlier Metric

- The metric we have used reflects the probability of the transaction sequence in the window. Higher the metric, lower the probability of the transaction sequence.

- Lower the probability of the transaction sequence, higher the likelihood of the transaction sequence being an outlier i.e., fraudulent.

- The size of the window holding the current transactions is an important factor. Smaller the time window more locally sensitive the result.

- Proper choice of the metric threshold is critical. A smaller value will cause more false positives and a larger value will cause more false negatives. Since false negatives are costlier, it’s more conservative to choose a smaller threshold value.
Some Storm Limitations

- *Storm* guarantees at least once message processing semantics. Some messages may be processed twice. Impact of this is not too serious in this case.

- Storm does not provide any state management features. We were doing state full processing inside the storm bolt with recent transactions stored in an in memory buffer.

- Impact of this could be serious in our case. A *Storm* node might go down right around when the buffer contains potentially fraudulent transactions.
Summing Up

- We have shown how different components of the Big Data ecosystem can be orchestrated to solve the real-time fraud detection problem.

- We have used Hadoop and Storm effectively as a solution platform. However, if I were to start the project today, I would seriously consider Spark and Spark Streaming.
Resources

- For more details, please visit my blog at http://pkghosh.wordpress.com/2013/10/21/real-time-fraud-detection-with-sequence-mining/

- The implementation is part of my OSS project on github at https://github.com/pranab/beymani

- A survey of different outlier detection algorithms is available in this blog post of mine http://pkghosh.wordpress.com/2012/01/02/fraudsters-outliers-and-big-data-2/