Using Deep Learning for optimized personalized card linked offer

March 5, 2018

Suqiang Song , Director, Chapter Leader of Data Engineering & AI
Shared Components and Security Solutions

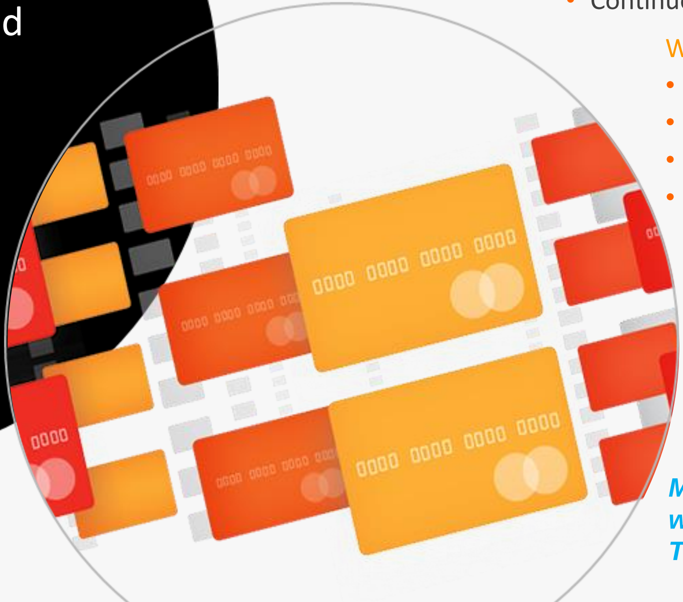
LinkedIn : https://www.linkedin.com/in/suqiang-song-72041716/

mastercard

Mastercard Big Data & AI Expertise

Differentiation starts with consumer insights from a massive worldwide payments network and our experience in data cleansing, analytics and modeling

What can
2.4 BILLION
Global Cards and
56 BILLION
Transactions/ Year mean to you?



MULTI-SOURCED

- **38MM+** merchant locations
- **22,000** issuers

CLEANSED, AGGREGATD, ANONYMOUS, AUGMENTED

- **1.5MM** automated rules
- Continuously tested

WAREHOUSED

- **10** petabytes
- **5+** year historic global view
- Rapid retrieval
- Above-and-beyond privacy protection and security

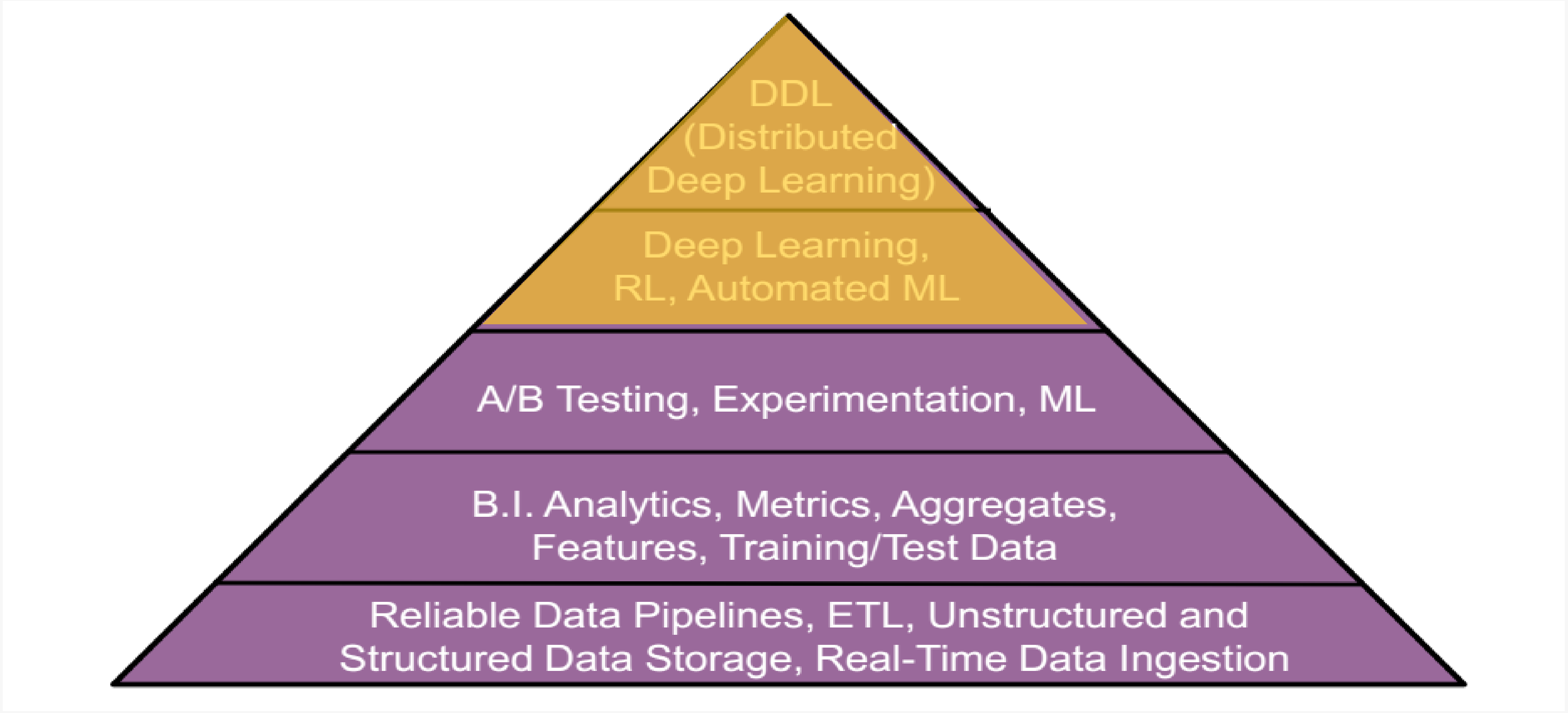
TRANSFORMED INTO ACTIONABLE INSIGHTS

- Reports, indexes, benchmarks
- Behavioral variables
- Models, scores, forecasting
- Econometrics

Mastercard Enhanced Artificial Intelligence Capability with the Acquisitions of Applied Predictive Technologies(2015) and Brighterion (2017)

Hierarchy of needs of AI

https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007?gi=b66e6ae1b71d


DDL (Distributed Deep Learning)
Deep Learning, RL, Automated ML
A/B Testing, Experimentation, ML
B.I. Analytics, Metrics, Aggregates, Features, Training/Test Data
Reliable Data Pipelines, ETL, Unstructured and Structured Data Storage, Real-Time Data Ingestion

Our Explore & Evaluation Journey

Enterprise Requirements for Deep Learning

Analyze a large amount of data on the same Big Data clusters where the data are stored (HDFS, HBase, Hive, etc.) **rather than move or duplicate data.**

Add deep learning capabilities to existing Analytic Applications and/or machine learning workflows **rather than rebuild all of them**.

Leverage existing Big Data clusters and deep learning workloads should be managed and monitored with other workloads (ETL, data warehouse, traditional ML etc..) **rather than separated deployment , resources allocation ,workloads management and enterprise monitoring**

……

Challenges and limitations to Production considering some "Super Stars"….

- *Claimed that the GPU computing are better than CPU which requires new hardware infrastructure (very long timeline normally)*
- *Success requires many engineer-hours (Impossible to Install a Tensor Flow Cluster at STAGE …)*
- *Low level APIs with steep learning curve (Where is your PHD degree ?)*
- *Not well integrated with other enterprise tools and need data movements (couldn't leverage the existing ETL, data warehousing and other analytic relevant data pipelines, technologies and tool sets. And it is also a big challenge to make duplicate data pipelines and data copy to the capacity and performance.)*
- *Tedious and fragile to distribute computations (less monitoring)*
- *The concerns of Enterprise Maturity and InfoSec (use GPU cluster with Tensor Flow from Google Cloud)*

……………

Maybe not your story , but we have ....

mastercard

What does Spark offer?

Integrations with existing DL libraries

- Deep Learning Pipelines (from Databricks)
- Caffe (CaffeOnSpark)
- Keras (Elephas)
- mxnet
- Paddle
- TensorFlow (TensorFlow on Spark, TensorFrames)
- CNTK (mmlspark)

Implementations of DL on Spark

- BigDL
- DeepDist
- DeepLearning4J
- SparkCL
- SparkNet

Need more break down …..

	Programming interface	Contributors	commits
BigDL	Scala & Python	50	2221
TensorflowOnSpark	Python	9	257
Databricks/tensor	Python	9	185
Databricks/spark-deep-learning	Python	8	51

Statistics collected on Mar 5th, 2018

Tensor Flow-on-Spark (or Caffe-on-Spark) uses Spark executors (tasks) to launch Tensor Flow/Caffe instances in the cluster; however, the distributed deep learning (e.g., training, tuning and prediction) are performed outside of Spark (across multiple Tensor Flow or Caffe instances).

(1) As a results, Tensor Flow/Caffe still runs on specialized HW (such as GPU servers interconnected by InfiniBand), and the Open MP implementations in Tensor Flow/Caffe conflicts with the JVM threading in Spark (resulting in lower performance).

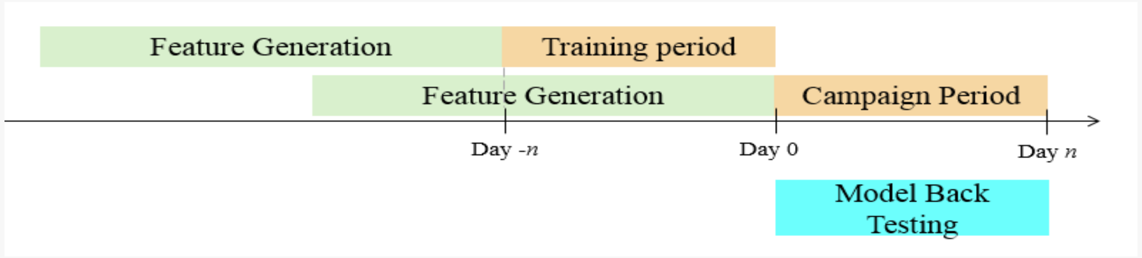
(2) In addition, in this case Tensor Flow/Caffe can only interact the rest of the analytics pipelines in a very coarse-grained fashion (running as standalone jobs outside of the pipeline, and using HDFS files as job input and output).

Use Case and POC

Business Use Case

Business driven :

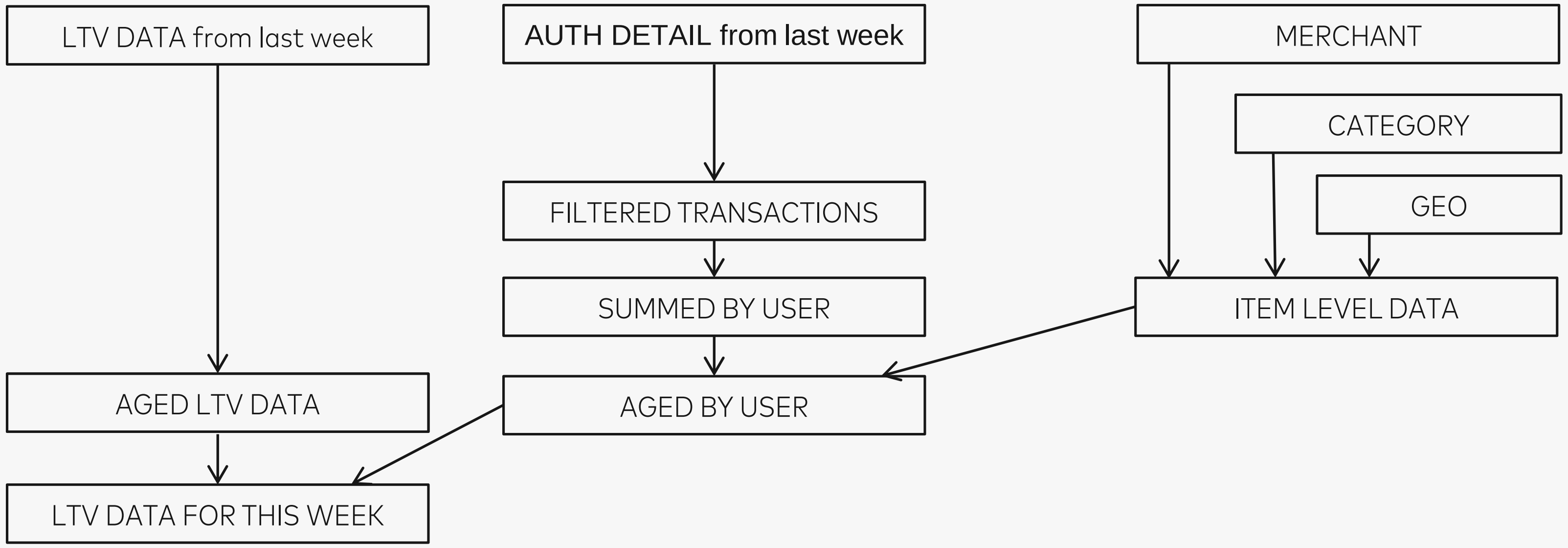
There are a variety of goals that advertising campaigns are targeted offering, like creating awareness or re-targeting consumers. For example, the latter could involve estimating the propensity of consumers to shop target merchants within several days. We take PCLO (Personal Card Lined Offers) use cases as our running example and focus on predicting users-merchants probability.



Goal :

implement new Users-Items Propensity Models with deep learning algorithms base on Intel BigDL framework to help our Loyalty solution to improve the quality , performance and accuracy of offer and campaigns design, targeting offer matching and linking .

Issues with Traditional ML : Feature Engineering Bottlenecks

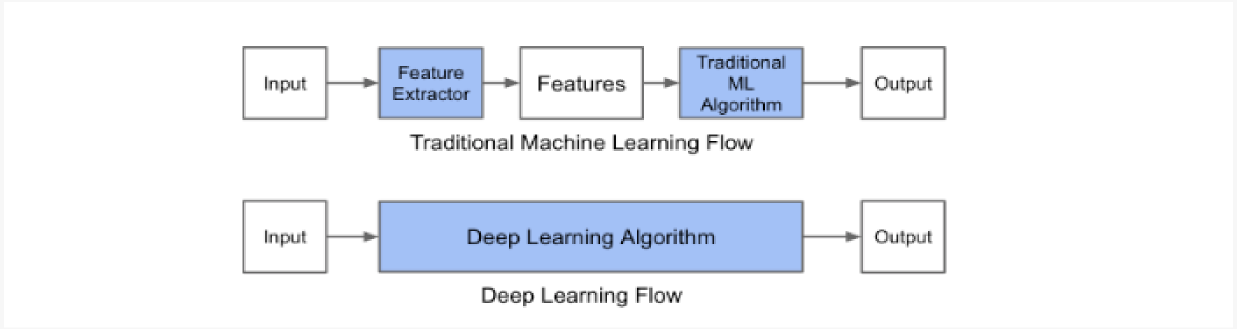


Bottlenecks

- Need to pre-calculate 126 Long Term Variables for each user, such as total spends /visits for merchants list, category list divided by week, months and years
- The computation time for LTV features took 70% of the data processing time for the whole lifecycle and occupied lots of resources which had huge impact to other critical workloads.
- Miss the feature selection optimizations which could save the data engineering efforts a lot

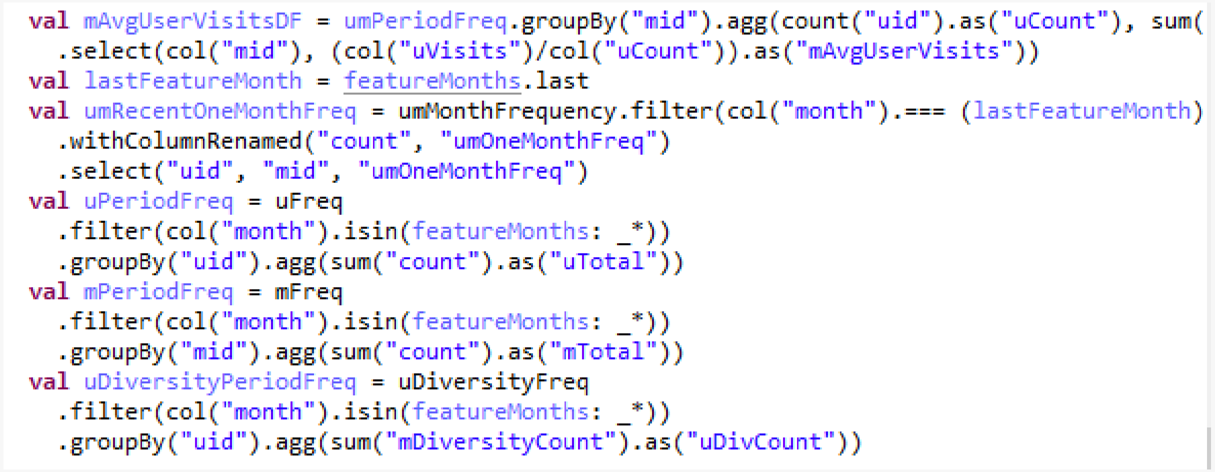
mastercard

With Deep Learning : Remove lots of LTV workloads and simply the feature engineering



Traditional Machine Learning Flow

Deep Learning Flow



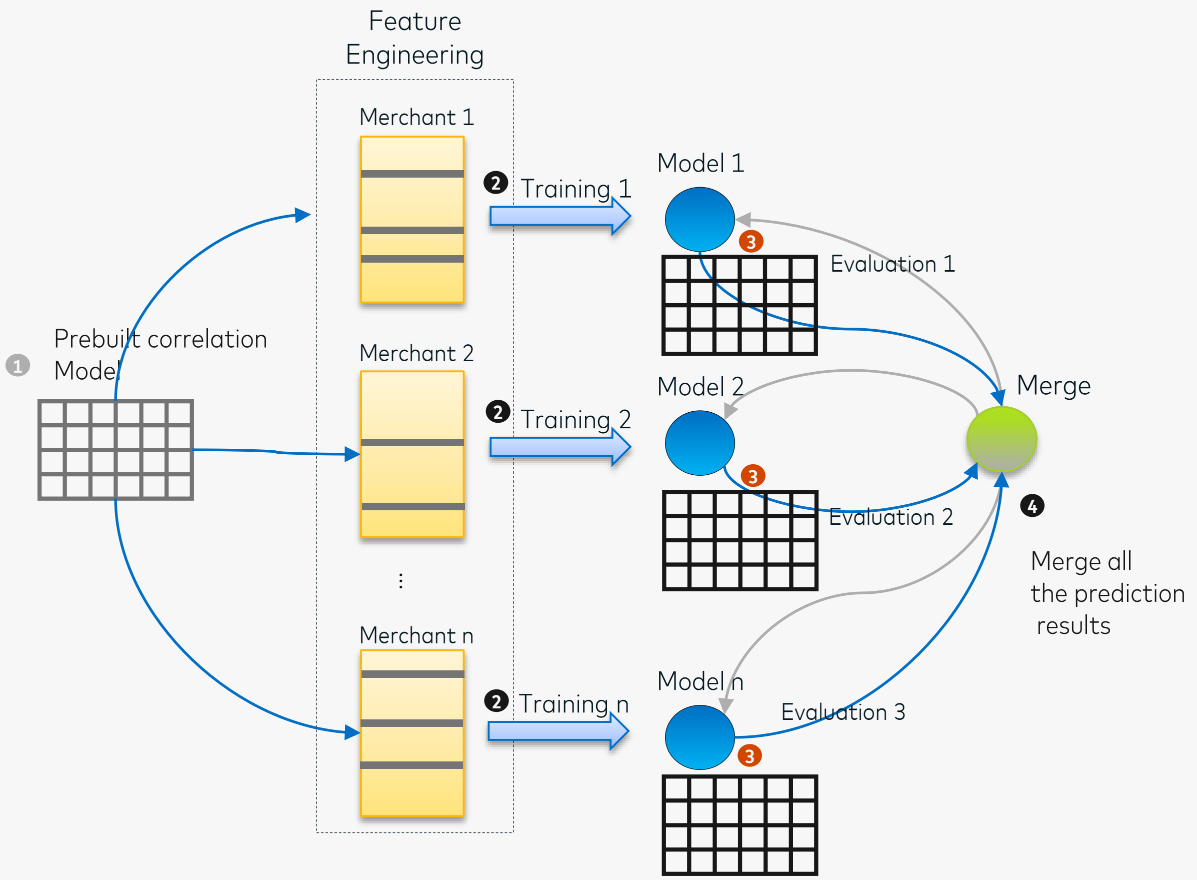
```
val mAvgUserVisitsDF = umPeriodFreq.groupBy("mid").agg(count("uid").as("uCount"), sum(
  .select(col("mid"), (col("uVisits")/col("uCount")).as("mAvgUserVisits"))
val lastFeatureMonth = featureMonths.last
val umRecentOneMonthFreq = umMonthFrequency.filter(col("month").=== (lastFeatureMonth)
  .withColumnRenamed("count", "umOneMonthFreq")
  .select("uid", "mid", "umOneMonthFreq")
val uPeriodFreq = uFreq
  .filter(col("month").isin(featureMonths: _*))
  .groupBy("uid").agg(sum("count").as("uTotal"))
val mPeriodFreq = mFreq
  .filter(col("month").isin(featureMonths: _*))
  .groupBy("mid").agg(sum("count").as("mTotal"))
val uDiversityPeriodFreq = uDiversityFreq
  .filter(col("month").isin(featureMonths: _*))
  .groupBy("uid").agg(sum("mDiversityCount").as("uDivCount"))
```

Improvements

- When build model , only focus on few pre-defined sliding features and custom overlap features (the users only need to identify the columns names from SQL source)
- Remove the LTV pre-calculations works a lot, saved 1~ 2 hours time and lots of resources
- The feature engineering can be constructed as templates or SQL query input , much simpler than complex ETL jobs
- BigDL (with neutral networks) will do the internal features selections and optimization automatically when it does cross validation at training stage

mastercard

Issues with Traditional ML : Model pipeline limitations

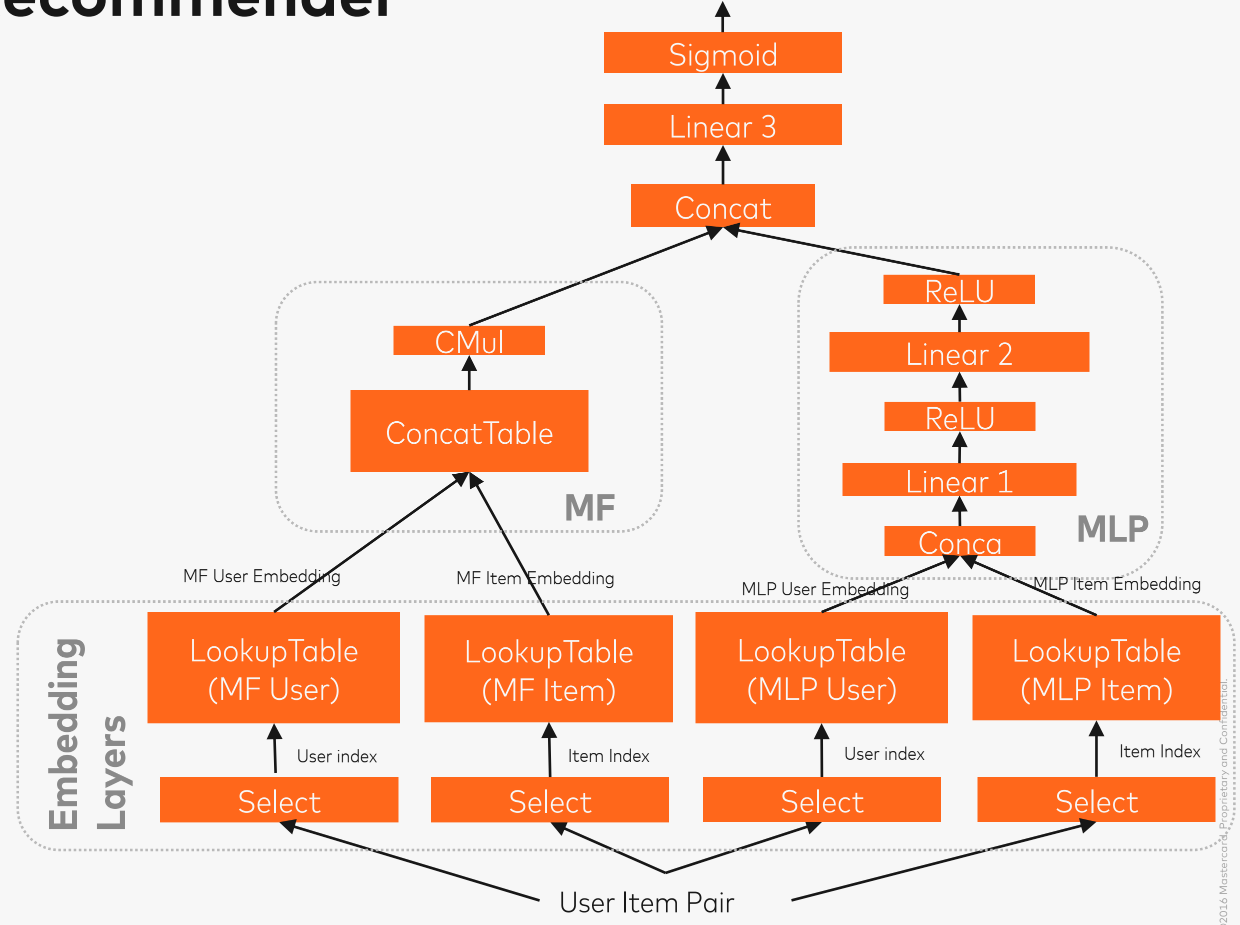


Limitations

- All the pipelines separated by merchants and generate one model for each merchant
- Have to pre-calculate the correlation matrix
- Lots of redundant duplications and computations at feature engineering ,training and testing process
- Run merchants in parallel and occupied most of cluster resources when executed
- Bad metrics for merchants with few transactions
- It was fine to support up to 200 merchants but couldn't scale to more

Approach: Neural Recommender

- **NCF**
- Scenario： Neural Collaborative Filtering ,recommend products to customers (priority is to recommend to active users) according to customers' past history activities.
- https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf
- **Wide & Deep learning**
- Scenario: jointly trained wide linear models and deep neural networks---to combine the benefits of memorization and generalization for recommender systems.
- https://pdfs.semanticscholar.org/aa9d/39e938c84a867ddf2a8cabc575ffba27b721.pdf



Benchmark approaches

Try two kinds of approaches to predict users-merchants (offers) propensities model.

- Traditional machine learning with handcrafted features (Based on Spark Mlib ALS)
- NCF & WAD learning with automated selected features

For each benchmark, evaluate and compare the regular performance metrics of ML models as follows:

- Receiver Operating Characteristic area under curve (ROC AUC)
- Precision Recall area under curve (PR AUC)
- Precision & Recall (https://en.wikipedia.org/wiki/Precision_and_recall)
- TOP 20 Precision

The major goals of this POC including:

- After couple rounds of benchmarks, verify and validate the differences, enhancements of deep learning approach compared with traditional machine learning.
- Abstract and purify the general data pipelines for similar kinds of modeling and data science engineering
- Learn the details of Intel BigDL, including APIs and mainstream models, master how to use it to handle the deep learning projects based on different business problems, domain models and data sets.

Environments

Select data sets

We selected 3 years credit transactions (from a Mid Bank), with those characteristics:

- Distinct qualified consumer : 675 k
- Distinct known transactions :1.4 billions (53 G for raw data)
- Target Merchants (Offers or Campaigns) for benchmark :2000

We choose 12 ~24 months for training and 1~2 months for back test.

Cluster (Prod Like Stage)

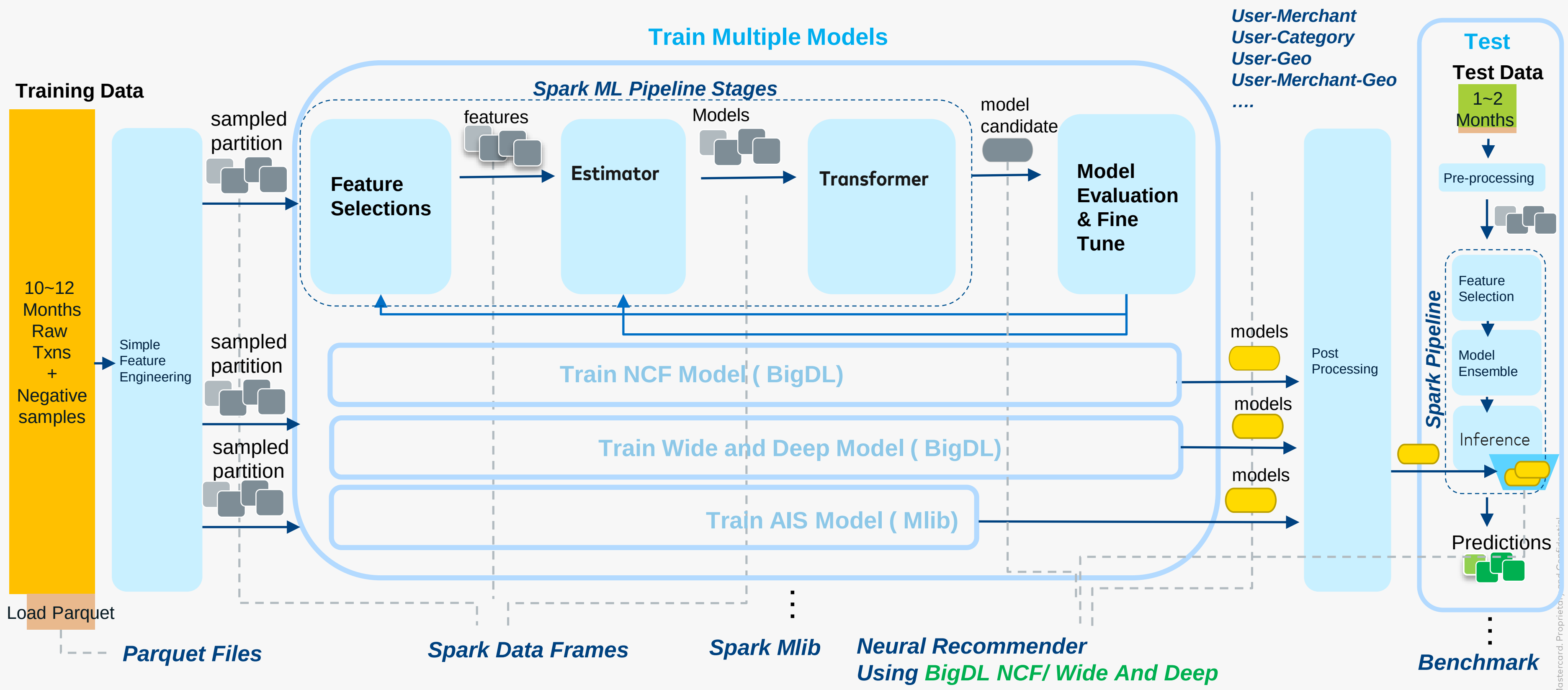
We selected a Stage Hadoop cluster which has same restrictions as PROD:

- 6 nodes cluster (3 HMN nodes, 3 HDP nodes), for each single node is physical box
- 24 hyper cores, 384 G memory, 21 T disk
- Hadoop distribution: CDH 5.12.1
- Spark version: 2.2
- Jdk 1.8

ML Libs

- Spark Mlib ALS (2.2)
- Intel BigDL (0.3)

Implementation : run BigDL & ALS over Spark on Hadoop



Benchmark results (> 100 rounds)

Mlib AIS

AUROC: A
AUPRCs: B
recall: C
precision: D
20 precision: E

```
Parameters :
MaxIter(100)
RegParam(0.01)
Rank(200)
Alpha(0.01)
```

BigDL NCF

AUROC: A+23%
AUPRCs: B+31%
recall: C+18%
precision: D+47%
20 precision: E+51%

```
Parameters :
MaxEpoch(10)
learningRate(3e-2)
learningRateDecay(3e-7)
uOutput(100)
mOutput(200)
batchSize(1.6 M)
```

BigDL WAD

AUROC: A+20% (3 % down)
AUPRCs: B+30% (1% down)
recall: C+12% (4 % down)
precision: D+49% (2 % up)
20 precision: E+54% (3% up)

```
Parameters :
MaxEpoch(10)
learningRate(1e-2)
learningRateDecay(1e-7)
uOutput(100)
mOutput(200)
batchSize(0.6 M)
```

PROD on-boarding journey and Tips for BigDL

Additional PROD on-boarding efforts

***Fine Tuning** (Among all the users and items , business wants to enhanced deep learning on subsets of them, such as improve the metrics for 50 pre-selected merchants from 6000 merchants)*

• Prepare training and testing datasets for pre-selected items
• Run the global training first , base on the phase 1 model , run fine turning by enhanced parameters with relevant train and test datasets from pre-selected items
• Update the model and generate separate evaluation matrixes.

***Continuous Learning**(only re-run the whole pipeline with incremental changed datasets such as daily changed transactions and benchmark the models)*

• Refresh the dimensional datasets (such as adding new users , items …)
• Load the history model to the context and update incremental parts of model based on the incremental data sets
• Periodic Re-training with a batch algorithm and time-series prediction
• Benchmark the history model and update model and on-board the better ones.

Additional PROD on-boarding efforts -- continue

Model Serving (Connect to existing business pipelines , offline ,streaming and real-time)

● Build the model serving capability by exporting model to scoring/prediction/recommendation services and integration points
● Integrate the model serving services inside the business pipelines , such as embed them into Map-reduce jobs for offline, Spark Streaming jobs for streaming , the real-time "dialogue" with Kafka messaging …

Cloud Native and API/Pipeline enablement

- Deploy to on premise and off-premise cloud native environments
- Microservice Implementation and deployment Strategy(Containerization & Container Orchestration)
- Automation API's to create autonomous deployment pipeline

Other PROD enablement Tips

Avoid OOM

- BigDL broadcasts model to each CPU core , not executor !
- Turn on spark.rdd.compress
- Tune shuffle a lot , such as spark.shuffle.manager=tungsten-sort
- Sparse Sensor , not Dense Sensor
- Tune GC a lot , god bless you ☺

Performance Tuning

- Batch Size matters
- Plan your epoch and iterations , tradeoff between performance and precision
- Separate data preparing ,training and Inference steps to multiple spark jobs
- Random sampling VS Invert sampling , another tradeoff

Some limitations should know

- Only support JavaSerializer ,issue with KyroSerializer
- BigDL 0.2 does not support Spark 2.x , so use 0.3 at least

Thanks

Q & A

mastercard