Implementation Guide

Big Data Analytics | Machine Learning on Intel® Platforms

Implementing End-to-End Predictive Analytics Solutions

Create a scalable, reusable machine-learning architecture based on Intel® technology and open source software that can solve real-world business problems in retail banking and other industries

Benefits of an FSI Recommendation System

A good recommendation system can help provide the following:

- Data about direct feedback from consumers relating to their satisfaction with products and services
- The ability to influence customers' buying decisions based on their needs
- A consistent customer experience that can help retain customers' trust and interest in services
- The ability to use the customer behavior data to drive business decisions

Table of Contents

Introduction 1
Accelerating Execution and Optimizations Using Intel® Technologies . . 2
Solution Overview 2
Retail Banking Use Cases for Predictive Analytics and Machine Learning........... 4
Recommendation Analytics Use Case Details.......................... 5
Performance 7
CDH for Predictive Analytics 7
Conclusion 8
Appendices.............................. 9

Introduction

With the right analytics tools, enterprises can grow their investment in data by making data-driven decisions and bridging the gap between cross-functional teams, data scientists, and software engineers. This implementation guide explains how to adopt machine-learning solutions to help your organization stay competitive and identify opportunities to optimize a big data solution.

End-to-End Optimized Machine-Learning Pipeline

The solution documented here demonstrates how foundational, reusable components can be used to develop an end-to-end machine-learning pipeline, using the Cloudera Distribution for Hadoop and Apache Spark. Additionally, it showcases how to tune and scale the solution to maximize utilization of the data center resources powered by Intel® products. These products include Intel® MKL, Intel® Turbo Boost Technology, Intel® HT Technology, and Intel® AVX-512.

Our testing revealed that end-to-end optimization and Intel MKL resulted in an **overall 21.4 percent performance boost** compared to not using Intel MKL, as measured by total time.[1] For the model training stage in particular, Intel MKL **accelerated the ALS algorithm up to 2.37x** (depending on the value of the `rank` parameter).[1]

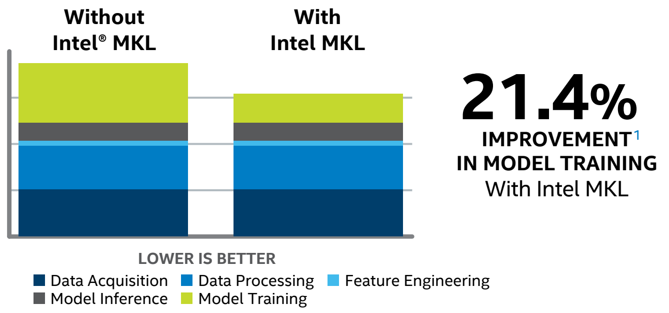
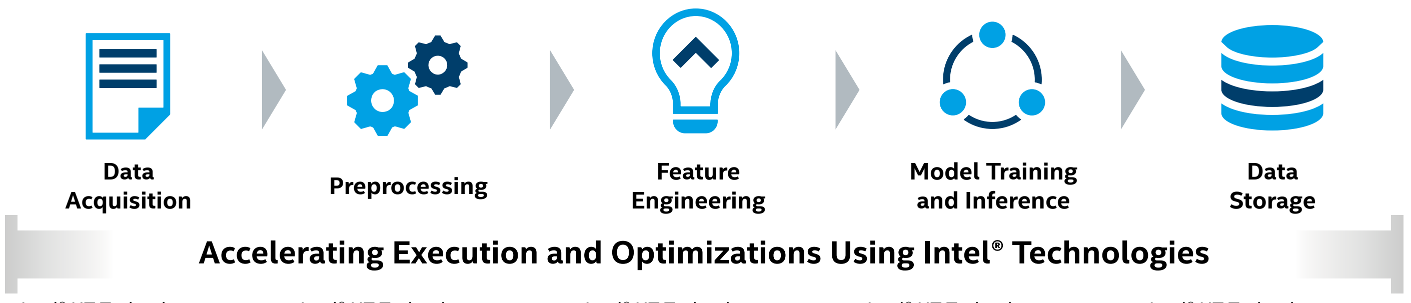


Figure 1. Intel® MKL plus end-to-end optimization improves machine-learning workload performance.

Machine Learning in the Financial Services Industry

The use cases discussed in this paper are drawn from retail consumer banking. However, the methodology shown in this reference architecture can be applied to many other use cases such as consumer retail, insurance, health and life sciences, manufacturing, energy, and transportation. Retail banks use financial technology to engage with their customers and expand their operations. Financial services organizations can use analytics solutions to recommend a variety of products and services to consumers. These include a new checking or saving account, credit cards, and lending services. The latter could be for automobiles, homes, or education, or refinance packages with attractive rates. Other services include asset and investment management (stocks and bonds) and international banking. The major challenge faced by banks—or any industry—is clearly understanding customers' needs and their choices, and using that knowledge to drive business and influence customers' buying decisions.


Data Acquisition
Preprocessing
Feature Engineering
Model Training and Inference
Data Storage
Accelerating Execution and Optimizations Using Intel® Technologies

Data Acquisition	Preprocessing	Feature Engineering	Model Training and Inference	Data Storage
• Intel® HT Technology • Intel® Turbo Boost Technology • Intel® QAT for data compression • Intel® AES-NI for fast encryption	• Intel® HT Technology • Intel® Turbo Boost Technology • Intel® QAT for data compression • Intel® Ethernet Adapters for 25 GbE/40 GbE networking • Intel® AES-NI for fast encryption	• Intel® HT Technology • Intel® Turbo Boost Technology • Intel® MKL acceleration • NVMe-based Intel® SSDs for fast shuffles	• Intel® HT Technology • Intel® Turbo Boost Technology • Intel® MKL acceleration • Intel® AVX-512 instructions	• Intel® HT Technology • Intel® Turbo Boost Technology • NVMe-based Intel® SSDs for optimized and reliable storage • Intel® Optane™ SSDs for low latency

Figure 2. Intel® technology can accelerate machine-learning workloads using built-in features of the Intel® Xeon® processor Scalable family and other optimizations such as Intel® MKL.

Accelerating Execution and Optimizations Using Intel® Technologies

Intel provides technologies to accelerate the various phases in a machine-learning workflow. Figure 2 shows some of the key aspects that can take advantage of optimized hardware and software integrated with Apache Hadoop. Effectiveness may vary based on workflow and features, and necessary benchmarks should be run before and after to measure the improvements in performance.

Solution Overview

This document describes a reference architecture for deploying a predictive analytics solution for retail banking. The end-to-end machine-learning pipeline is based on the CDH platform on Intel® architecture and adjacent technologies. Figure 3 provides an overview of the building blocks for this solution architecture. Each data engineering or analytics stage defines its API, which helps data scientists and data engineers to standardize the data exchanges across multiple use cases.

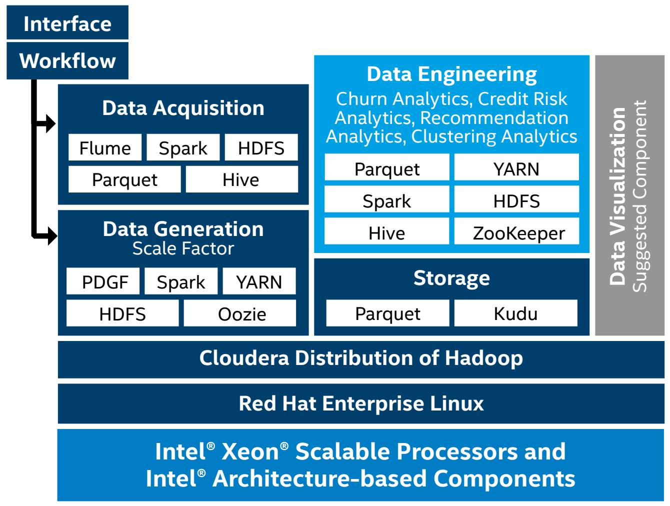


Figure 3. Our machine-learning reference architecture is built on Intel® technologies and CDH and uses many open source tools to support several predictive analytics use cases.

The predictive analytics workload for retail banking includes the following modules:

- **Workflow** represents the pipeline execution sequence, dependencies, and data flow, including the configuration files, workload input parameters, and storage paths (see Figure 4).

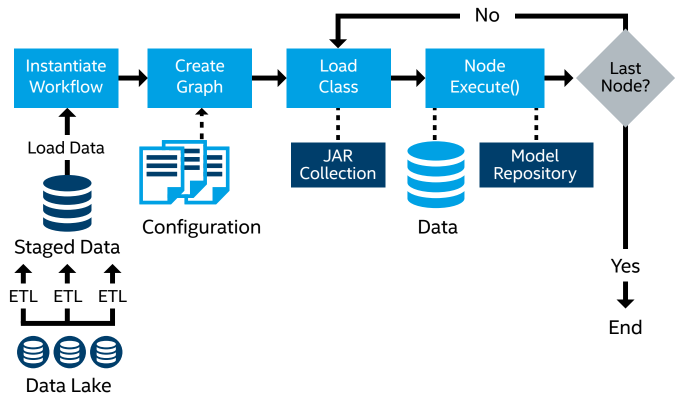


Figure 4. Workflow for creating and managing the data flow in a machine-learning pipeline.

- **Interface** provides the abstraction for machine-learning stages or nodes in the pipeline implementation.
- **Data source** represents a subset of data collection from multiple internal and external data sources in a banking environment and third-party vendors (see Figure 5).
- **Data acquisition** for data storage and management of the staging dataset (performs ETL operations on Hadoop Distributed File System [HDFS]).
- **Advanced data pipelines** for predictive analytics use cases using the banking data model. Each use case can configure one or more data engineering stages: preprocessing, feature engineering, model training, and model inference. See the sidebar Sample Predictive Analytics Use Cases.
- **Model storage** to store and manage the model metadata, model type, parameters, and files after the feature engineering and/or model training stages in the pipeline. The model and metadata both are captured for future analysis.
- **Data storage** to store intermediate results during the workflow, including reading and writing to tables.

The intermediate staging results are stored on HDFS in Parquet format, managed by Hive; The final prediction results are stored on Apache Kudu for fast access to verify the quality of predictions.

The interface and workflow modules make the machine-learning pipeline more abstract, decoupled, and reusable with modularized configurations to control the data flow. For example, the submodules (preprocessing, feature engineering, model training, and model inference) allow a data scientist to modify a specific implementation of the submodule and quickly plug the changes into pipeline as desired. This solution includes the complete reference binaries and the dataset.

Sample Predictive Analytics Use Cases

The four sample use cases for retail banking are as follows:

Recommendation analytics. Recommendation systems in banking and finance help guide users or customers to discover new products based on implicit feedback from other similar customers, their activities, and preferences relating to the services that they use currently. We discuss the implementation and performance for this use case in greater detail in this guide.

Customer analytics for predicting customer churn. Churn prediction analytics helps to identify customers who might stop using banking services or might be inclined to switch to competitors. We treat the analysis as a binary classification model (just two classes) where the customer will either churn or stay. If the prediction value is true, then there is a high likelihood that the customer might stop using some or all the services.

Customer analytics for predicting credit risk. Credit risk analytics helps to identify clients or potential applications that might be high-risk given their financial portfolio and historical transactions. Usually, credit risk is computed using a variety of external factors such as credit scores, balances across all financial assets, and the debt products owned by the customer. However, to simplify our use case, we assume that all information has already been gathered by the data acquisition engine and is part of the banking data model. To extend the source to other data points, appropriate changes should be made to the EDW steps such as integration with other third-party services.

Customer segmentation analytics for user behavior modeling. One of the many requirements of a financial services provider is to better understand the categories of customers so that the provider can model its services while maximizing target penetration. This type of modeling is often achieved using clustering and identifying the similarities or associations in the available data. Once the clusters have been identified, necessary actions or strategies can be developed for each segment.

Note: Only recommendation analytics is discussed in detail in this paper.

Simplified Retail Banking Data Model

Typically, banks implement a proprietary system to meet their custom needs for processing transactions, creating reports, or handling customer queries. Common examples of such systems include the Oracle Financial Services Data Warehouse and the SAP Business Warehouse for banking and financial services. Several other similar solutions are available, as well. Figure 5 provides a high-level logical depiction of the physical data warehouse implementation for retail banking. Note that Figure 5 is an illustrative example only; a full-fledged banking data model will be far more complex and is outside the scope of this paper.

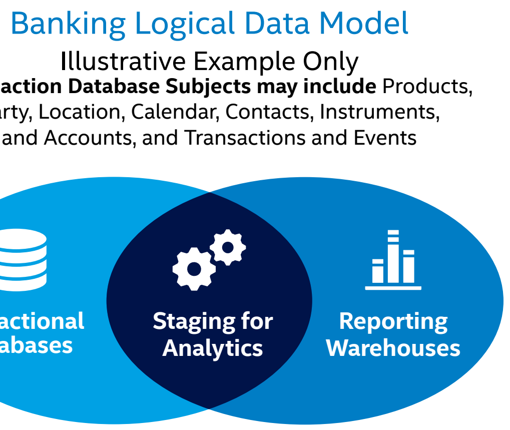

Banking Logical Data Model
Illustrative Example Only
Transaction Database Subjects may include Products, Party, Location, Calendar, Contacts, Instruments, and Accounts, and Transactions and Events
Transactional Databases
Staging for Analytics
Reporting Warehouses

Figure 5. The retail banking logical data model consists of several transactional databases, a staging area for analytics (often called the enterprise data warehouse or EDW), and reporting warehouses.

Overall, the logical data model consists of the following:

- **Transactional databases.** These stores are typically updated very frequently as new events occur. Enterprises use relational stores to implement these data models. This space is mature enough to accommodate lower latencies and to scale both horizontally and vertically to handle higher throughput.
- **Reporting warehouses.** These stores are typically created for reporting purposes and are accessed on an on-demand basis. The report generation is usually a batch process running on a periodic schedule such as nightly, weekly, monthly, quarterly, or yearly. Typically, enterprises model these as large-scale data integration jobs that can be horizontally scaled and do not have any latency constraints. Traditional business-intelligence and data-integration frameworks, such as Informatica Data Integration Hub and Oracle Fusion, act as intermediaries to orchestrate these jobs. Recently, enterprises have started using MapReduce-based frameworks such as Apache Hadoop and Spark to make these processes more robust and fault-tolerant while being able to scale horizontally.
- **Staging for analytics.** A middle ground exists where necessary data is either staged as-is or preprocessed and aggregated for further decisions based on analytics. In this guide, we focus on this staging warehouse and refer to it as the EDW. This staging area is typically updated on a schedule so that analytical jobs are run in a cadence that provides meaningful results in the time frame during which the data was collected. The EDW is also shared among many data scientists and application developers, so they most likely use only certain tables from this store.

Figure 6 illustrates a banking EDW model that shows relationships between different fact tables such as Product, Loans, and Credit Cards. (Note that a real-world EDW model would be far more complex.) Various combinations of these tables are used by data scientists and application developers. For example, the *Product Order* table acts as the central table, which typically captures a product that the bank offers, contains a link to the account with which it is associated (a client could have multiple accounts), and the client ID (for aggregation and analytical purposes).

In Figure 6, variable names in italics indicate the foreign keys. The arrows represent the foreign key relationship between tables. Variable names in bold are the target fields in the machine-learning pipeline that we are trying to predict and are used as ground truth (to measure the accuracy of the training set's classification for supervised learning techniques). These target variables for the use cases are as follows:

- The recommendation analytics target is an implicit rating based on the *Product Order* table.
- The customer churn analytics target is account_status from the *Account* table.
- The customer credit risk analytics target is risk from the *Client* table.
- The customer segmentation analytics target is based on review entered in the *Client* table.

Retail Banking Use Cases for Predictive Analytics and Machine Learning

The banking data model shown in Figure 5 provides a solid environment for using machine-learning and analytical techniques to derive insights from transactional and customer data.

Figure 7 illustrates three suggestive pipelines that benefit from machine learning and analytics: a recommendation analytics pipeline, a customer analytics pipeline, and a customer segmentation analytics pipeline. The customer analytics pipeline supports several use cases including churn analytics and credit risk analytics. In the figure, the highlighted boxes represent where Intel MKL can be used to accelerate performance.

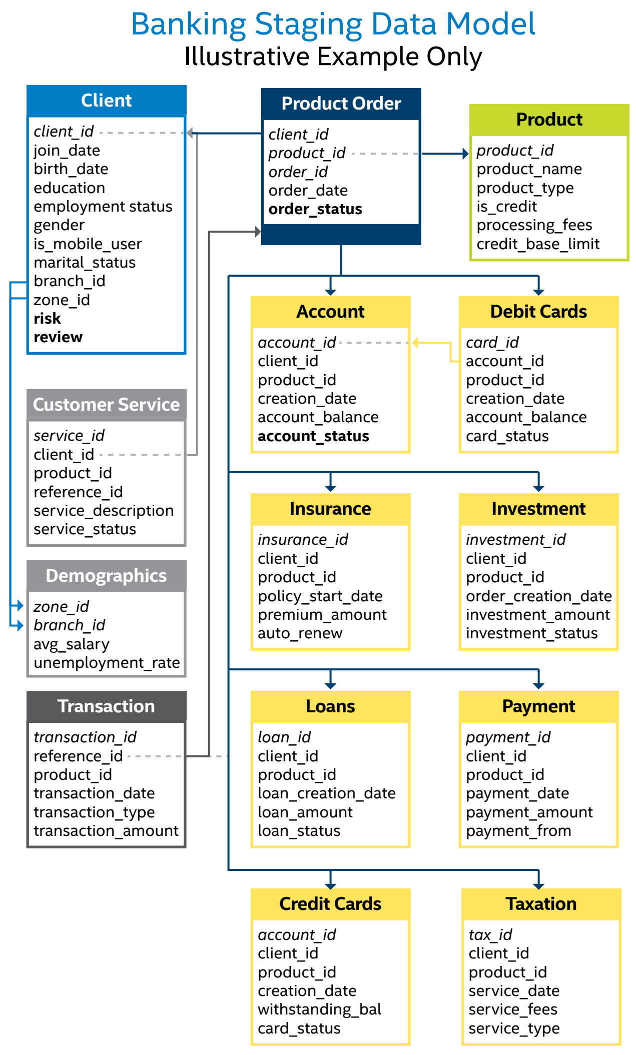


Figure 6. The typical retail banking EDW consists of many transactional databases and tables.

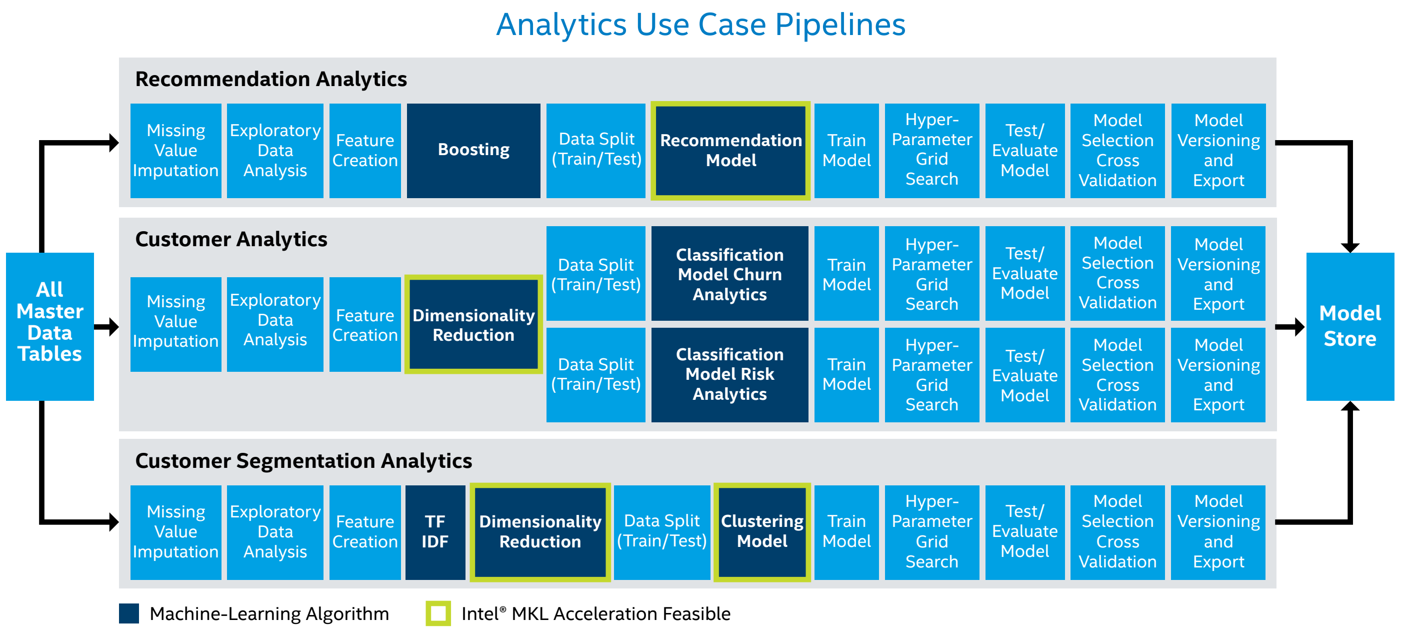

Analytics Use Case Pipelines
All Master Data Tables
Recommendation Analytics
Missing Value Imputation
Exploratory Data Analysis
Feature Creation
Boosting
Data Split (Train/Test)
Recommendation Model
Train Model
Hyper-Parameter Grid Search
Test/ Evaluate Model
Model Selection Cross Validation
Model Versioning and Export
Customer Analytics
Missing Value Imputation
Exploratory Data Analysis
Feature Creation
Dimensionality Reduction
Data Split (Train/Test)
Classification Model Churn Analytics
Train Model
Hyper-Parameter Grid Search
Test/ Evaluate Model
Model Selection Cross Validation
Model Versioning and Export
Data Split (Train/Test)
Classification Model Risk Analytics
Train Model
Hyper-Parameter Grid Search
Test/ Evaluate Model
Model Selection Cross Validation
Model Versioning and Export
Customer Segmentation Analytics
Missing Value Imputation
Exploratory Data Analysis
Feature Creation
TF IDF
Dimensionality Reduction
Data Split (Train/Test)
Clustering Model
Train Model
Hyper-Parameter Grid Search
Test/ Evaluate Model
Model Selection Cross Validation
Model Versioning and Export
Model Store
Machine-Learning Algorithm
Intel® MKL Acceleration Feasible

Figure 7. Several retail banking use cases can benefit from machine learning.

We discuss only the recommendation analytics use case in detail in this paper. The following section provides more information about the individual stages in data exploration and data science adoption of the pipeline.

Recommendation Analytics Use Case Details

The following sections describe the dataset used for testing and discuss the step-by-step machine-learning stages in the recommendation pipeline—from acquiring data and preprocessing to model training and inference.

Dataset

The data for the recommendation system is obtained from several tables (as shown in Figure 6), to understand the customers' financial portfolio and daily interactions.

- The *Client* table records the customer enrollment and demographics information.
- The *Product* table comprises information regarding products and different types.
- The *Account* and *Investment* tables capture the different accounts like savings, checking, premium services, and enlists the accrued invested funds towards retirement and wealth plans.
- The *Credit Card* and *Loan* tables track the outstanding amount on active credit and loans accounts.
- The *Transaction* and *Payment* tables maintain the aggregated amount for the flow of money between different clients, accounts, and card services.
- The *Taxation*, *Insurance*, and *Customer services* tables capture the other miscellaneous services occasionally provided to clients.

We used a highly customizable third-party data generation tool, Parallel Data Generation Framework (PDGF), to simulate the reference banking data. It is based on the probability distribution adapted from the census income dataset and models some of the client demographic information such as marital status, education level, gender, capital gains, and work hours per week.

Data Acquisition and Preprocessing

Many enterprises have faced the challenge of maintaining legacy MapReduce code base. Hive provides flexibility to quickly adapt Spark processing for analysts/developers with SQL background. Hive on Spark (HoS) is more suitable for complex workloads with multiple MapReduce stages involving shuffle, union, and join operations. In contrast, Hive on MapReduce (HoMR) can suffice for queries with a small number of stages. We recommend using HoS for better performance.

Using the HDFS command-line interface or Flume, data is loaded into HDFS, a fault-tolerant replicated distributed filesystem. We then load the dataset using Hive in Parquet format. We also provide options for other formats like TextFile, CSV, and ORC. Hive provides SQL-like familiarity and makes it easy to manage and analyze the data on HDFS. Parquet provides columnar storage with compression and encoding for efficient ETL operations. CDH v6 supports vectorization on Intel architecture for Parquet and ORC formats. In vectorized query execution, data rows are batched together and represented as a set of column vectors. Cloudera recommends using Parquet for optimizations using vectorization because of the available support for Parquet across other components. Hive also provides scalability, redundancy, and extensibility. In our experiment, we have observed **up to 2.8x compression for Parquet** compared to the TextFile format for Hive tables (see Appendix C: Data Acquisition – Compression Benefits for Hive on Spark with Parquet Format).

Upon acquisition, we identify the different tables and relevant features useful in model training, change the data types, and then join the aforementioned tables to create staging tables with tuples (client_id, product_id, aggregated_features, and so on), grouped together to retrieve aggregated features resulting from joining different tables.[2] It's not plausible to get direct feedback like product ratings from clients for each product consumed by them. Therefore, the rating needs to be derived from customers' interaction with different accounts, products, investments, and card services. These aggregated features are later used during feature engineering for computing the feedback on respective products (see "Feature Engineering"). The relative latest activity should be given more weight in computing the feedback for respective products. To consolidate data across all clients and products, we created the following denormalized tables:

- The *Product Order* table summarizes purchased products by the customer over years of business with a bank.
- The *Recommendation Ratings* table filters for the last X years to adjust the training dataset size.

The initial value for a product's base rating is solely based on when the product was purchased (that is, order placement data). The resulting columns after joins are checked for null values and substituted with default values. We recommend storing the intermediate results at the end of each stage of the workflow and deleting those intermediate results after successful completion of the next stage. This may help avoid re-runs due to errors in a module and may also ease debugging issues in the data flow between processes and verification of module functionality during runtime. The data acquisition and preprocessing steps include performing complex queries including union, joins, sum, count, average, filter, and windowed operations on Parquet tables managed by Hive.

We recommend tuning the Hive configuration during the data acquisition and data preprocessing, modifying parameter values based on the scale factor. To learn more about detailed configuration and tuning, refer to the Hive or Cloudera documentation.

Feature Engineering

The feature engineering module selects the data to be fed into the model training step. Typically, dimensionality reduction, boosting, regularization, one-hot encoding, indexing, and normalization are the most common steps data scientists use to prepare the data before it is ready for model training.

For product recommendation analytics, we compute delta ratings as an effect of customers' behavior (such as purchasing activity) from different tables to influence the estimates of the base rating as mentioned previously in the preprocessing stage. The base rating is further augmented based on client demographics and product type. For example, demographics information like *is_home_owner*, *employment_status*, *annual_income*, *is_mobile_user*, *mobile_alerts_on*, *customer_type*, and *is_social_profile_connected* are used to influence rating values for products designed for relevant types of customers. The historical transactions with different products (such as number of credit cards, saving accounts, loans, seasonal promotions for new accounts for students and taxation services) are used to boost the respective product's rating. This changes the ratings from explicit to implicit feedback, because we use other features to internally compute the rating value. For more information on explicit versus implicit feedback, refer to the discussion on the spark.org site.

Additionally, the various delta ratings are computed from the other tables and added up to the final product rating for model training. The MaxAbsScaler is used to measure the weight of features like number of active accounts, sum of invested amount, number of active credit cards, outstanding balance in loan accounts, and volume of payment and transaction amounts across all clients and products. Spark ML has a dedicated feature package (org.apache.spark.ml.feature*)* that provides useful classes and functions to transform the features to simplify the effort for data scientists. For example, VectorAssembler, StandardScaler, Normalizer, Bucketizer, OneHotEncoder, StringIndexer, and Tokenizer are commonly used.

Model Training and Cross-Validation

Collaborative filtering and content-based filtering are among the frequently used algorithms for recommendation systems. We use the Alternating Least Squares (ALS) implementation of collaborative filtering to learn the latent factors necessary to build our recommender.

USING ALS[3]

For clients (*client_id*) and products (*product_id*), the algorithm starts by creating an c *x p* matrix R where r is the computed rating value (*product_rating*) for each product (p) and each client (c):

$$R_{cp} = \left\{ \begin{matrix} r & \text{if client c ordered product p} \\ 0 & \text{if client c did not order product p} \end{matrix} \right\}$$

$$R_{cp} = \mathrm{CP}^{T}$$

The goal is to factorize the rating matrix R into two: matrix C (client factor) and matrix P (product factor), using a small number k, such that C summarizes each client c by k vectors and P summarizes each product p by a k-dimensional vector. For each iteration, the algorithm first optimizes P by fixing C and, then fixes P and optimizes C, for the given rating matrix R_{cp}. We repeat this optimization cycle for a defined maximum number of iterations. The cycle must be tuned to reach a convergence point where the iterative changes in the two matrices are quite small.

IMPROVING THE ALS MODEL

We use Spark ML Pipelines to build a PipelineModel or CrossValidatorModel that runs across several regularization parameters and performs k-fold cross-validation. All these parameters are configurable for data scientists to run different hypotheses. For comparing models, we use RegressionEvaluator, which uses Root Mean Squared Error (RMSE) as the metric to evaluate ALSModel. Finally, we package the algorithm model, grid for parameter tuning, and model evaluator into the stages of the PipelineModel (or CrossValidatorModel if cross-validation is enabled).

For improving model performance, we have incorporated various parameters from the ALSModel class. We set `ImplicitPrefs` as true, to indicate that the rating value is a derived value and not readily available from customers. The confidence in this derivation can be controlled by setting the value for `Alpha`, with a default value of 1. The `rank k` parameter represents the number of latent factors and is set to 40 in the default configuration file. We found that tuning the number of partitions as a multiple of the total number of cores available on all workers is most useful in increasing the parallelism for Spark jobs. To improve performance in the distributed ALS algorithm, we experimented with intermediate user and item factors (that is, `ProductBlocks` and `UserBlocks`.) These factors control the number of blocks that are cached in memory during runtime by the ALSModel. This helps to reduce the shuffle of feature vectors between workers. Refer to Spark ML ALS documentation for more details.

Model Storage

The output of the CrossValidatorModel or PipelineModel is stored on HDFS and a corresponding entry is input into the "model_catalog" table in Kudu, which is managed by the Model Storage module. The CrossValidatorModel returns the best model from the different k-fold validations executed on the training dataset. The cross-validator internally represents the best model as well as the metrics that helped it determine how the model was selected. The model catalog includes model version, timestamp, parameters, model path, and validation accuracy on the test dataset. These stored models and their metadata can enable decision makers to choose which model to load for inference, based on whether the business needs the model provisioning services to be more accurate or faster.

Model Inference

The solution offers different types of model selection policies from the model catalog for a particular use case. Users can specify their choice in the configuration as

- **Specific.** Uses a universally unique identifier (UUID) for the model to be loaded
- **Best.** Chooses the model with the lowest cost/error value
- **Latest.** Chooses the last trained model for that pipeline

The inference pipeline is similar to the training pipeline, except it uses the transform method on the trained model to obtain recommendations for the new set of clients and services. The inference job can be executed on demand or as a scheduled job on a batch of features. Scheduled jobs tend to optimally utilize cluster resources. The feature values are typically pre-computed and stored using batch inference jobs that run on a weekly or monthly cadence. The recommendations are sorted prediction scores of how likely the client is to subscribe to the service or purchase a product. Usually, we select the top k products (where k lies between 3 to 10) with the highest predicted ratings, and these recommendations are proposed to the clients as part of a marketing campaign. These prediction results are stored in datastores like Kudu for faster retrieval.

Model Drifts

Model training is a continuous process; without regular model training, the customer experience can degrade. Model accuracy can be affected by the rate of increase in the number of clients and products and a product's frequency of use and popularity. Over time, a model can succumb to model drift.[4] When customers pursue certain recommendations or update their ratings, it creates a feedback loop and requires the model to be re-trained on new data. It is possible to understand the impact of this feedback and analyze this information to prevent model drift. The frequency of model training depends on the business need and cost constraints; however, it is critical in some industries such as online retail services to stay competitive and relevant. Enhancements suggested in the paper, "Deconvolving Feedback Loops in Recommender Systems" could be added to improve model accuracy.

Performance

In the recommendation pipeline, each stage exhibits slightly different workload characteristics, and requires optimization and tuning to maximize utilization of all cluster resources. For brevity, this guide discusses only two stages in detail: preprocessing and model training.

The preprocessing stage uses HoS aggregation queries, which are compute-intensive tasks and include union, joins, aggregate, and filter operations. We recommend tuning the garbage collection activity and using vectorization for Hive queries. These queries scale linearly with additional executor and nodes.

The model training stage is memory-intensive and requires fine-tuning Spark memory configurations. We focused mainly on tuning the parameter `rank` with respect to the evaluation metric Root Mean Square Error (RMSE). The `rank` parameter indicates intermediate matrix-rank of the Client and Product matrices that are decomposed from the rating matrix by the ALS algorithm. The increase in rank impacts the complexity of solving the two matrices by increasing the number of matrix operations. This behavior illustrates the need for more memory to perform in-memory caching for Client and Product matrices. The Intel MKL parcel for Cloudera boosts machine-learning and data analytics performance by accelerating BLAS Level 1 routines and functions (examples include `dot`, `scal`, and `daxpy`) in Spark MLlib. In this pipeline, the Intel MKL parcel for Cloudera accelerates the ALS algorithm up to 2.37x for Rank 160 (see Figure 8).

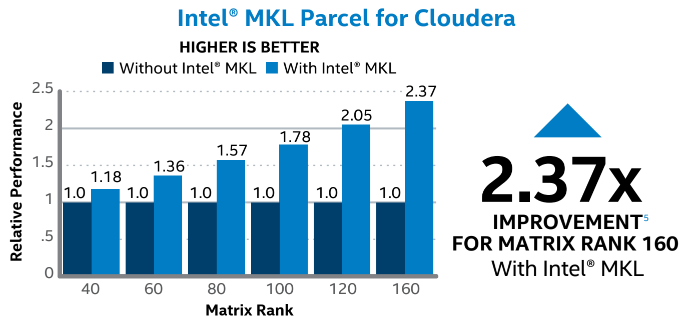


Figure 8. Intel® MKL Acceleration for model training using ALS algorithm (18 worker nodes) SF 2000.

Analyzing the end-to-end pipeline runtime with consideration for the best model deployment resulted in a 21.4 percent increase in overall performance, compared to a cluster without Intel MKL.[5] Intel MKL acceleration can enable data scientists to experiment with other parameters like `alpha`, regularization parameters, and cross-validation to fine-tune the model.[6]

Next Steps

In future tests, we plan to extend our performance testing by adding Intel® Optane™ persistent memory to the configuration for running model training with larger `rank` values. We can also analyze and optimize for the following scenarios, creating dedicated resource pools for model training and the rest of data engineering operations:

- Tuning resources for serving multiple parallel queries
- Running multiple use cases

CDH for Predictive Analytics

Cloudera Distribution for Hadoop (CDH) is an open source platform that is built entirely on open standards. CDH includes robust Apache Hadoop 3.0 and Spark 2.x, along with a variety of other open source components from the big data ecosystem. This reference architecture uses CDH's scalable platform to deploy our machine-learning pipelines and analytics with the banking data model. Table 1 lists elements of the open source ecosystem that are used in this reference architecture. The system requirements and Cloudera role distribution are listed in Appendix A: System Requirements and Appendix B: Cloudera Role Distribution - 21 Nodes. Detailed best practices for cluster sizing can be found here.

Table 1. CDH Open Source Components for Advanced Analytics

Analytics Stage	Open Source Tools
Data Acquisition	HDFS, Hive, Flume[a], Kafka[a], and Scoop[a]
Data Processing	Apache Hive on Spark, Apache Spark and Structured Streaming[a], Hive on Tez[a]
Model Training	Apache Spark and Spark ML
Data Storage	• **Apache Parquet:** the column-oriented data serialization standard for efficient data analytics • **Apache ORC:** self-describing type-aware columnar file format designed for Hadoop workloads • **Apache Kudu:** a new, scalable and distributed table-based storage for hybrid architectures that handle both transactional and analytics workloads
Model Storage	Apache Parquet, Apache Kudu

[a] Components can be added; currently not included in the reference architecture.

Apache Spark Optimization

Apache Spark is well integrated with CDH and widely used for developing analytics pipelines. Spark is easy to use, expressive, and optimized to achieve high throughput. Apache Spark 2 is equipped with data serialization, whole-stage code generation, and an improved catalyst optimizer. The Intel MKL parcel for Cloudera accelerates Apache Spark MLlib by optimizing the low-level routines for performing common linear algebra operations. Examples of such operations include vector addition, scalar multiplication, dot products, linear combinations, and matrix multiplication. **No application code changes are required** to benefit from Intel MKL acceleration and integration is simple using the Intel MKL parcel for Cloudera.

Conclusion

Providing a customer-centric experience, with product and service recommendations that are tailored to specific customer preferences and behaviors, is an important aspect of doing business in today's digital world. Intelligent recommendation systems, driven by machine-learning algorithms such as ALS are important. But just as vital as the actual recommendations is the algorithm's overall performance—customers expect near-real-time recommendations. Our research has revealed that a real-world machine-learning infrastructure consists of more than just the machine-learning code. Lack of attention to the other areas of data analytics (such as storage, BLAS routine performance, and memory configuration) can sabotage machine-learning projects.[7] For machine learning to deliver on its promise, it must be based on a solid data engineering foundation. Specifically, our research demonstrates that Parquet vectorization for data processing can increase data compression by 2.8x[8] and Intel MKL acceleration for Spark ML speeds up model training by up to 2x.[9]

Authors

- **Snehal Adsule**, Cloud Solutions Engineer, Data Platforms Group, Intel Corporation
- **Shibani Singh**, Cloud Solutions Engineer, Data Platforms Group, Intel Corporation
- **Rodrigo Escobar**, Cloud Solutions Engineer, Data Platforms Group, Intel Corporation
- **Swepna Doss**, Cloud Solutions Engineer, Data Platforms Group, Intel Corporation
- **Amandeep Raina**, Cloud Solutions Engineer, Data Platforms Group, Intel Corporation

Find the solution that is right for your organization. Contact your Intel representative or visit **Intel® Analytics**.

References

- White Paper: Hidden Technical Debt in Machine Learning Systems
- Cloudera: Intel® MKL for Cloudera
- Intel: Accelerate Machine-Learning Workloads with Math Kernel Library Performance Brief
- Faster Swarms of Data: Accelerating Hive Queries with Parquet Vectorization

Learn More

You may find the following resources helpful:

- Machine Learning-Based Advanced Analytics Using Intel® Technology Reference Architecture
- Intel's Machine Learning webpage
- Intel® Xeon® Scalable Processors
- Intel® QuickAssist Technology
- Intel® Optane™ persistent memory
- Intel® Ethernet Adapters
- Intel® Solid State Drives
- Intel® Select Solutions

Appendix A: System Requirements

Table A1 provides the hardware and software requirements for this reference architecture.

Table A1. Bill of Materials

Hardware Requirements	
Processor	21x Intel® Xeon® Gold 6248 processor (20 cores, 40 threads, 2 sockets)
Memory	384 GB or higher (12x 32 GB, 2933 MHz DDR4, DIMMs)
Boot Drive	1x 960 GB Intel® SSD DC S4500 Series SSDSC2KB96
Storage for HDFS	8x 4 TB SEAGATE ST4000NM0095 for 32 TB storage
Storage for Yarn temporary	2x 2 TB Intel® SSD DC P4510 for 4 TB storage
Data Network	Intel® Ethernet Network Adapter X722 (10 GbE)
Intel® Hyper-Threading Technology	Enabled
Intel® Technology	Enabled
Power-Management Settings	Performance
Software Requirements	
• Red Hat Enterprise Linux CentOS Linux v7.6	• Hive 2.1.1-cdh6.2.0
• OpenJDK 1.8	• Apache Kudu 1.9.0-cdh6.2.0
• CDH v6.2.0	• PDGF v2.6 (Parallel Data Generation Framework)
• Apache Spark 2.4.0-cdh6.2.0	• Intel® MKL parcel (mkl-2019.5.281)
• Hadoop 3.0.0-cdh6.2.0	• Intel MKL Wrapper Parcel 1.0

Appendix B: Cloudera Role Distribution - 21 Nodes

Table B1 provides information about how the Cloudera roles are distributed across the 21 nodes.

Table B1. Cloudera Role Distribution

Utility Nodes (3)			Worker Nodes (18)
Master Node #1: • Name Node • YARN Resource Manager • ZooKeeper #1 • Job History Server • Spark History Server • Kudu Master	**Utility Node #2:** • Cloudera Manager • Cloudera Manager Management Service • ZooKeeper #2 • Secondary Name Node	**Utility Node #3:** • HiveServer2 • Hive Metastore • Oozie • ZooKeeper #3 • HDFS Balancer • Hue Server	• Data Node • Node Manager • Hive Gateway • Spark Gateway • Kudu tablet server (3 out of 18)

Appendix C: Data Acquisition – Compression Benefits for Hive on Spark with Parquet Format

Our results indicate that the Parquet format conserves considerable storage space—up to 2.8x less total storage space compared to the TextFile format.

Table C1. Compression Ratios for TextFile versus Parquet Formats

Database Tables	**Sizing Formula** Client Factor (CF) = 1000 Scale Factor (SF) = 2000	**Raw Data Size** TextFile Format	**HDFS Data Size** Parquet Format	**Compression Ratio** TextFile: Parquet
Account	1000 * CF * SF	159.9 GB	68.7 GB	2.33
Client	100 * CF * SF	99 GB	78.4 GB	1.26
Credit card	300 * CF * SF	49.2 GB	15.6 GB	3.15
Customer service	10 * CF * SF	1.1 GB	372.7 MB	3.02
Debit card	100 * CF * SF	49 GB	16.9 GB	2.9
Demographics	10 * SF	736.5 KB	420.8 KB	1.75
Insurance	200 * CF * SF	26.8 GB	15 GB	1.79
Investment	200 * CF * SF	21 GB	11.1 GB	1.89
Loan	400 * CF * SF	53.6 GB	24.8 GB	2.16
Payment	200 * CF * SF	103.7 GB	66.2 GB	1.57
Product	60	3.5 KB	4.6 KB	0.76
Taxation	10 * CF * SF	1018.3 MB	425.8 MB	2.39
Transaction and Transaction Loan[a]	25000 * CF * SF 400 * CF *SF * (Loan Duration[a])	6.8 TB	2.2 TB	3.09
Total		**7.3 TB**	**2.6 TB**	**2.8**

[a] The Transaction Loan table is distributed based on active years, loan duration, and payment schedule for a loan account.

[1] Testing by Intel as of June 24-27, 2020. 21-node cluster. 2x Intel® Xeon® Gold 6248 processor @ 2.50 GHz, Intel® Hyper-threading Technology ON, Intel® Turbo Boost Technology ON, OS = CentOS Linux release 7.6.1810 (Core), Total Memory = 384 GB (12 slots/16 GB/2933 MHz), BIOS = SE5C620.86B.02.01.0008.031920191559 (ucode: 0x500002c). Software Stack: Cloudera Distribution for Hadoop (CDH) v6.2.0, Apache Spark v2.4.0-cdh6.2.0, Hadoop v3.0.0-cdh6.2.0, Hive v2.1.1-cdh6.2.0, Kudu v1.9.0-cdh6.2.0, Parallel Data Generation Framework (PDGF) v2.6. Intel MKL Enablement: CDH Parcels :mkl-2019.5.281,mkl wrapper parcel 1.0

[2] A tuple is a sequence (or ordered list) of n elements.

[3] More explanation on ALS implementation can be found at the following links:
"Large-Scale Parallel Collaborative Filtering for the Netflix Prize," link.springer.com/chapter/10.1007/978-3-540-68880-8_32
"Speeding up Distributed Big Data Recommendation in Spark," arxiv.org/pdf/1508.03110.pdf

[4] Model drift, sometimes called concept drift, is when the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions become less accurate as time passes.

[5] See endnote 1.

[6] See endnote 1 for configuration details.

[7] "Hidden Technical Debt in Machine Learning Systems," papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

[8] See Appendix C: Data Acquisition – Compression Benefits for Hive on Spark with Parquet Format.

[9] See endnote 1.