Samantha - the Engine of Data-Driven Intelligent Systems

Version 1.0.0, Qian@GroupLens

Introduction

- what's Samantha
    - A generic recommender and predictor server. Machine learning is the core, but much more than that, particularly for the purpose of recommendation/ranking/candidate generation/evaluation.
    - MIT licence, oriented to production use (online field experiments in research and typical industrial use)
- the amazing features of Samantha, what Samantha can do in simple words: full-fledged, self-contained server that can be used in production right away with one configuration file
    - data management, including offline and online, in (indexing) and out (post-processing), bootstrap once for all
    - model management, abstracted model operations
    - data processing pipeline, data expanding and feature extraction framework
    - state-of-the-art models: collaborative filtering, matrix factorization, knn, trees, boosting and reinforcement
    - experimental framework for A/B and bucket testing
    - feedback (for online learning/optimization) and evaluation (for experimenting) loops among application front-end, application back-end server and Samantha
    - abstracted model parameter server, i.e. extensible variable and index spaces
    - generic oracle-based optimization framework/solver with classic solvers
    - flexible model dependency, e.g. model ensemble, stacking, boosting
    - schedulers for regular model rebuilding or backup
    - control and customize all these components through one centralized configuration file
- why do I develop Samantha
    - user-centered/human-centered research
    - offline to online trend
    - lack of good ones, many but no ideal
- the targeted audience/users of Samantha (users in this doc refer to those who use Samantha; instead, end users is used to refer to the ultimate users of applications built using Samantha)
    - individuals/organizations who want to deploy a data-driven predictive system with minimum effort. They might need it to support answering relevant research questions involving an intelligent predictive part in their system or just to have an initial try to see the effects of such a predictive component.
    - individuals/organizations who are working on comparing and developing new machine learning or recommendation models/algorithms, especially those who care about deploying their models/algorithms into production and evaluate them in front of end users
- the goal
    - the engine of data-driven intelligent systems

- what to support in the near future
  - text modeling
  - model chaining/composition, i.e. deep learning
  - scale up to clusters: learning algorithm and offline data storage

Related Work

- overall, three perspectives regarding the differences
  - recommender toolkit → recommender server
  - machine learning library → machine learning system
  - offline → online, research/testing → production
- compare with recommender tools, lenskit, mahout, libfm etc.
- compare with machine learning tools, weka, libsvm, scikit-learn etc.
- compare with machine learning systems: xgboost, mxnet, etc.
- compare with machine learning server prediction.io

Dive into Samantha

Easy/Minimum Setup (*play in activator*)

- Samantha is an application of play framework (which itself is a HTTP web application framework). It manages library dependencies and building through SBT (http://www.scala-sbt.org/).
- You can easily set up Samantha by the following three commands (assuming Linux-like systems)
  - git clone https://github.com/grouplens/Samantha.git
  - cd server && ./activator -jvm-debug 9999 -Dhttp.port=9000 run
  - go to your browser and access http://localhost:9000
- If you see the following screen (Figure 2) then you've successfully set up a development mode Samantha server
- you can use development mode Samantha server for testing purpose or research projects involving only offline data sets, e.g. evaluating different recommendation/machine learning models/algorithms with offline data sets.

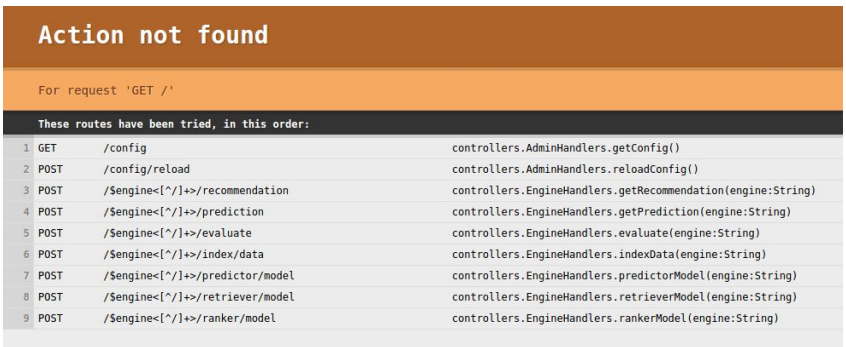

Figure 2. All HTTP interfaces that are supported by Samantha in a browser screen.

Start Using Samantha: Engines and Key Components

- Engine and its Components are the first concepts users need to know to use Samantha.
- There are two configured engines ready for use after you clone Samantha source. One is an example showing the minimal configuration you need to set up an engine in order to test it out and start using it; Another one is an example showing a minimal configuration to have a production ready/deployable recommender server (using MovieLens as an example here). While reading the following introduction, you can refer to these two configuration files to have a better idea on what those concepts look like.
- Samantha has two types of engines: Recommender vs. Predictor engine.
    - Predictor engine is designed for applications which only need scoring a list of items, entities or events etc. For example, spam filtering is a perfect example in which the application only needs a predictive system to tell it whether a message is a spam or not after actually receiving the message. On the other hand, Recommender engine is designed for applications which need a selection of items or a ranked list of things. For example, almost all product/item recommender systems fit in this type of engine. However, the definition of Recommender engine in Samantha is broader than the normal known recommender systems because it only specifies that the output should be a ranked list of things. For example, the scenario that an application asks for the most similar words to a given word also perfectly fits in Recommender engine. Another example is conversational agents in which the best sentence or sentences are picked when the agents are interacting with end users.
    - A Recommender engine includes all functionalities of a Predictor engine. To note, predictors in a Recommender engine are also directly exposed to outside, which means you can directly ask for predictions from a Recommender engine.
    - A Recommender engine typically relies on Predictors to generate a ranked list. For example, a typical rating-based recommender uses a rating predictor to generate predicted ratings for items and then rank them based on the predictions.
- You can have as many engines as you like in Samantha. They are functionally separated from each other like different domains (although there are several caveats you need to be aware of mentioned later). You are free to use engines to scope your application requirements. A recommended scenario is to scope making prediction or recommendation on different types of items into different engines for the purpose of optimizing them separately, for example using one engine for recommending products and another engine for recommending people/friends etc, similarly for Predictor engines. Of course, you always have the choice of deploying multiple Samantha servers as you like which could be a good fit for different teams or products.
- An engine consists of multiple types of engine components. Recommender engine has more components than Predictor engine. Each engine can have as many number of different types of components as you need. They are differentiated by their names which can be any string given by users.
- Predictor engine component

    - Indexer: this type of component deals with data indexing in real-time when receiving data from applications.
        - Samantha supports many classes of indexer implementations, typically corresponding to different data storage backend systems. For example, Elasticsearch has its corresponding indexer, so does Redis. They are all NoSQL databases in comparison with SQL-based (schema-based) ones like MySQL or PostgreSQL. Samantha uses them extensively because of their flexibility and performance.
        - You can have multiple indexers with the same class by differentiating them with names. Typically, one indexer is configured for one data type, much similar to a database table, although the indexer class is usually general enough to take in any data types with different data fields. In production, it is recommended to use one indexer for each data type the application sends in because it makes the integration between Samantha and the application much easier and clearer. This will be elaborated in details later.
    - Predictor:
    - Router
    - Evaluator
    - Scheduler
- additional Recommender engine component
    - Recommender
    - Retriever: this type of component is responsible for retrieving candidates for a Recommender engine.
        - Recommender systems research typically focus on predictive model and algorithm innovation and evaluation and mostly ignore the problem of retrieving in an actual recommender system. Partially, it's because there are not many online experiments reported in so much details in academic and industrial research as to go into this aspect. Another reason is that this aspect is very engineering intensive and hence may not be interesting in academic research. However, most production systems have to deal with it because of performance issues, i.e. it is not possible to score all items when item space is big and especially when the predictive model involves complicated computation. Therefore, an initial candidate generation process is a necessary component for a Recommender engine.
        - Retriever can be simple and straightforward, e.g. retrieving all of the items or retrieving the top popular items. It can be complicated and critical to achieve good recommendations, too. For example, we can build simple machine learning models in a retriever to score all items and pick the top to generate candidates. Another possible approach is to build fast associative models as an item-based k-nearest neighbor algorithm does, in which candidates become those most similar items to a user's previously liked items. We can also blend multiple retrievers with different priorities, e.g. first using any results produced by a personalized retriever and resort to non-personalized one when necessary.

- Ranker

Interfaces and Request Processing Flow

Samantha's capabilities are provided to users through two types of HTTP interfaces/requests. This is the very front/boundary of Samantha, through which the outside systems or users interact with. One type is *management* requests, including model management and configuration management. The other type - *realtime indexing/serving* requests - provides recommendation or prediction and the capability of indexing data in real-time. Figure 1 illustrates the processing flow of Samantha receiving different types of requests.

1. global configuration management

GET /config
POST /config/reload

Samantha has a global configuration service (jvm wise, not cluster) referred to in this doc as Config Service. It manages all configurations (by distributing configurations into engine components which will be described later) including reading and loading configurations from a unified configuration file. These two configuration management requests are passed by handlers to Config Service directly. They are the two simplest requests.

2. model management

POST /$engine/predictor/model
POST /$engine/retriever/model
POST /$engine/ranker/model
POST /$engine/evaluate

3. recommendation and prediction serving

POST /$engine/recommendation
POST /$engine/prediction

4. data indexing

POST /$engine/index/data

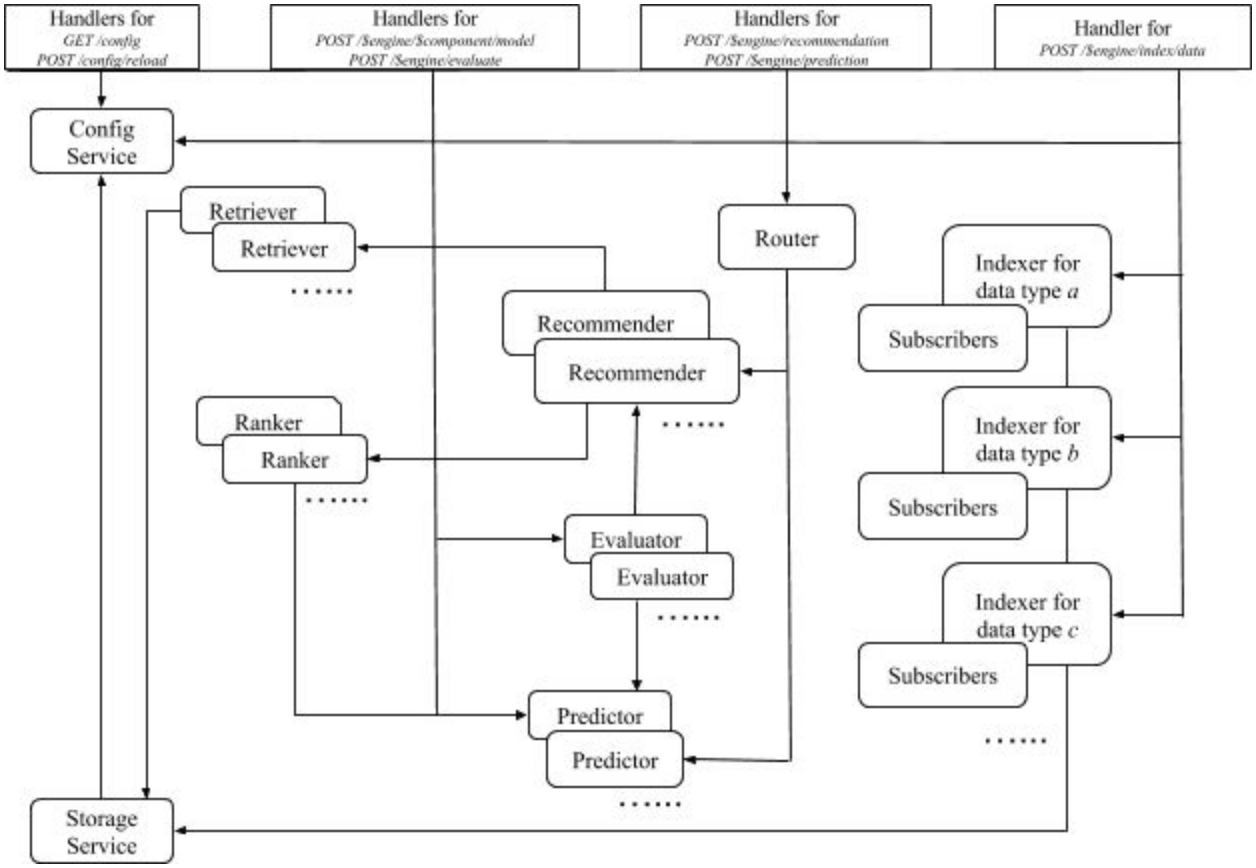

Figure 1. Samantha Data/Request Processing Flows. The directions of the arrows represent data flow or component dependencies.

Indexers for Data In/Out

- JSON, DAO
- real-time data and offline data indexing
- grouped data points output

Retriever for Candidate Generation

Learning Data, Learning Model and Optimization Solver

- oracle-based optimization framework
- parametric model
- loss function: normal pointwise loss and listwise ranking loss
- (online) optimization method using stochastic oracles

Light-weight Feature Extraction Framework

Heavy-weight Data Expanding

Model Service and Management

- model defaults

Flexible Component and Model Dependency

- Component Construction
    - system initiation phase
    - request serving phase
- model ensemble/stacking
- recommender depending on another recommender/reinforcement learning

Evaluator and A/B or Bucket Testing

- Metrics
- Experiment Group Assignment/Tagging

Online Learning and Evaluation Loops

- group tagging
- from data to (feature extracted) instance

Scheduler for Offline Jobs

Extensibility

Distributed Protocol Design and Lightweight Core

Shallow Dependencies/Flattened Software Layers

Extended Interface through Request

Scalability

Index/Variable Space (Parameter Server)

Extensible DAO

Redis and Elasticsearch Integration

Summary: Production Ready with One Configuration File

Case Studies/Examples

Model and Algorithm Research

- feature-level model ensemble
- model-level model ensemble
- boosting and GB-CENT

Application Integration and User Research

- main integration threads
    - prediction/recommendation request
    - data indexing request
- once-for-all bootstrap
- A/B or bucket testing (between-subjects assignment)

Integrating with Other Systems (Extension/Plugin)

- xgboost
- lenskit
- wikibrain

- etc.

Conclusion and Outlook

- what Samantha provides
- what's the roadmap

Acknowledgement

Appendix

More about Installation/Setup

Current Technology Stack

Knowledge Requirement to Use Samantha

- only interact with configuration file and deploying the system
- from application to data/computing model (previous data analysis work helps)
- understand the data processing flow of Samantha
- get familiar with the java api doc which shows how to use each type of components
- (may need to support mysql/postgresql)

Knowledge Requirement to Extend Samantha

- Design Patterns/Principles
    - philosophy of Samantha design: "minimum viable product"
    - dependency injection
        - strongly and weakly follow the pattern
        - inject as needed especially for services
        - actual working class and config class
    - singleton
    - server-oriented and be mindful of the running phase
        - system initiation
        - request serving
- java coding and debugging, play framework
    - configuration
    - dynamic loading and compiling
    - sbt/scala building
    - handlers vs. system logics
- http protocol, http server/service-oriented programming
- JSON

- MySQL, PostgreSQL, Redis and Elasticsearch
- Database and local file-based storage (extensible to distributed file storage, e.g. HDFS)

List of Models/Algorithms

Dive into Classes/Interfaces

References