Samantha - the Engine of Data-Driven Intelligent Systems

Version 0.1.0, 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)
- 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 bandits/reinforcement learning
    - 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
    - many options out there 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/predictive 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

Background

- terminologies
    - recommender systems vs. machine learning
    - models vs. algorithms
    - human theory vs. machine learning theory

Application/Product

- All the following applications can be fit into a common machine learning framework/theory because they can be modeled as a common decision making computational process. A generic system should be able to incorporate them straightforwardly
    - recommenders
        - personalized vs. most similar items
        - items can be broad e.g. people/friend/celebrity, movie/trailer/book, work requests
        - rating based or action based
        - tag based
        - location-aware
        - buddy recommender
    - (precision) crowdsourcing
    - content (quality) scoring, online community user retention
    - conversational agent (a recommender with sentences as the items although much more than this, the framework is the same)
    - search with queries/context, i.e. the core part of search engines
    - (computational) ads display
    - spam filtering, lead scoring, fraud detection
    - etc.

Statistical Machine Learning Theory

- supervised learning
    - minimize the error/loss of model prediction/decision with respect to an unknown (true) distribution

$$W^* = argmin_W E_P[L(f(W, X), Y)]$$

$$\text{the model is}: L(f(W, X), Y) \text{ and } P$$

    - instructive feedback
    - empirical risk minimization

$$W^* = argmin_W \sum_{n=1}^{N} L_n(f(W, X_n), Y_n)]$$

- reinforcement (online) learning:
    - maximize the reward of a policy (based on model prediction) cumulatively in the long run
    - or minimize the regret of a chosen policy compared with the theoretically best policy in certain hypothesis family with the respect to the true reward distribution in the environments
    - evaluative feedback
    - ...
- current primary model: supervised model with objective/response (model the decision making objective)
    - binary classification, regression -> multi classification -> ranking
- secondary model: i.e. unsupervised models
    - intermediate processes, commonly used as assistance (features) to the primary model, e.g. text embedding, entity embedding
- the trending primary model: reinforcement learning
- ***collaborative filtering***
    - item-based knn
        - model-free (statistical model) or model-based (human model) depending on the committee
        - X is the current userId and its history ratings and the current itemId and its previous ratings (the dimension is as big as the number of itemIds or userIds)
    - matrix factorization
        - X is the current userId and the current itemId
- learning algorithm, computational level:
    - cf knn: similarity computation
    - most of others: solving an optimization, batch/online

From Theory to Practice[1]

- model and loss function abstraction: $f(X)$ and L
- feature/representation computation: X
- optimization solvers to solve $argmin$ (this includes the hot domain deep learning)
- a computational pipeline (architecture) specially for machine learning (including the methods of collaborative filtering)

[1] http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf, particularly Rule #29, #31, #32, #33

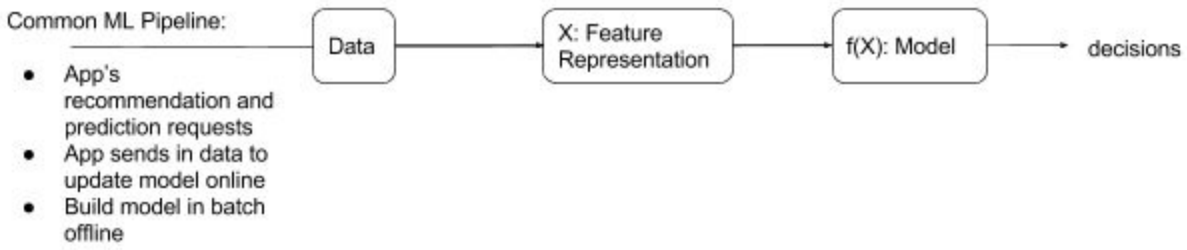

Figure 1. A diagram of common machine learning computation pipeline.

Related Work

- software: programming language -> libraries/APIs -> command line tool -> system -> server/service -> application
- overall, three perspectives regarding the differences
    - recommender toolkit → recommender server
    - machine learning library → machine learning system
    - offline → online, research/testing → production

	General Domain	**Software Type**
Lenskit	recommendation	general tool/library
MyMediaLite	recommendation	general tool
SLIMx	recommendation	specific tool
libsvm **libFM** **apex svdfeature**	machine learning	specific tool
scikit-learn **weka**	machine learning	general library
dmlc: **xgboost** **difacto**	machine learning	(distributed) system
dmlc: mxnet **theano** **caffee** **tensorflow**	deep(architectural/graphical) machine learning	general library, (distributed) system
sparkML (mahout)	machine learning	general library, (distributed) system
prediction.io	machine learning	server, (distributed) system

Samantha	machine learning recommendation	general library, server, (distributed) system

Table. A high-level comparison of the available softwares for building data-driven predictive systems.

Start Using 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.

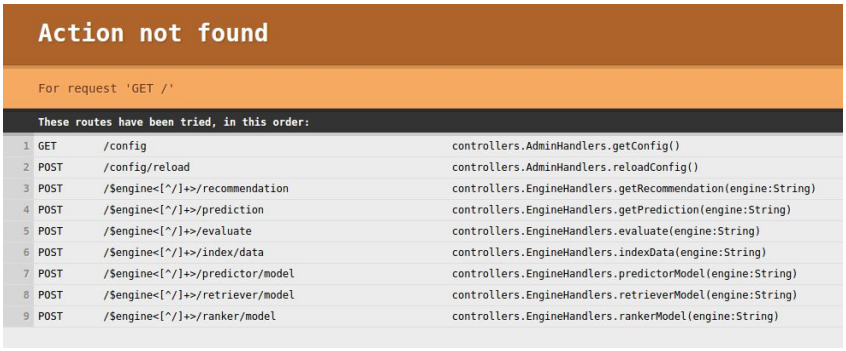

Figure 2. All HTTP interfaces that are supported by Samantha in a browser screen.

Engines and Components

- Engine and its Components are the first concepts users need to know to use Samantha. They are corresponding the key interfaces (Java term) in Samantha, which basically dictates the processing flow and framework.
- There are three configured engines ready for use after you clone Samantha source (currently: *qian/demo branch*).
    - One is an example showing the minimal configuration you need to set up an engine in order to start changing and using it: *server/conf/starter.conf*

    - Another one is an example showing a minimal configuration to have a production ready/deployable recommender server: *server/conf/movielens.conf* (using MovieLens as an example here).
    - The last one is specially created for demonstration purpose of this doc: *server/conf/movielens-demo.conf*.
    - While reading the following introduction, you can refer to these three 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***: this type of component is the core part of operationalizing machine learning theory as mentioned above, roughly falling into the supervised learning domain.
        - this is mainly a wrapper for a machine learning model/algorithm implementation, which is essential for providing Samantha the capability to integrate other machine learning libraries
    - ***Router***: this type of component implements how to find the right recommender/predictor for a request, which is the foundation of bucket (A/B) testing framework.
    - ***Evaluator***: there are two standard types of evaluators
        - Prediction evaluator: which provides the ability to compute prediction metrics for any predictor component
        - Recommendation evaluator: which can evaluate any recommender component by computing top N ranking metrics.
    - ***Scheduler***: this type of component enables regular model and data maintenance.
- ***Additional Recommender engine component***
    - ***Recommender***: in Samantha, a standard recommender is just the combination of a retriever and ranker, i.e. a retriever retrieves initial candidates from the storage or an initial model and feeds them into ranker to generate the ranked list
    - ***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 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***: this type of component takes in an initial candidate item set and rank them based on certain criterion which could just be the output/score of a Predictor component.

HTTP 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 3 illustrates the processing flow of Samantha receiving different types of requests.

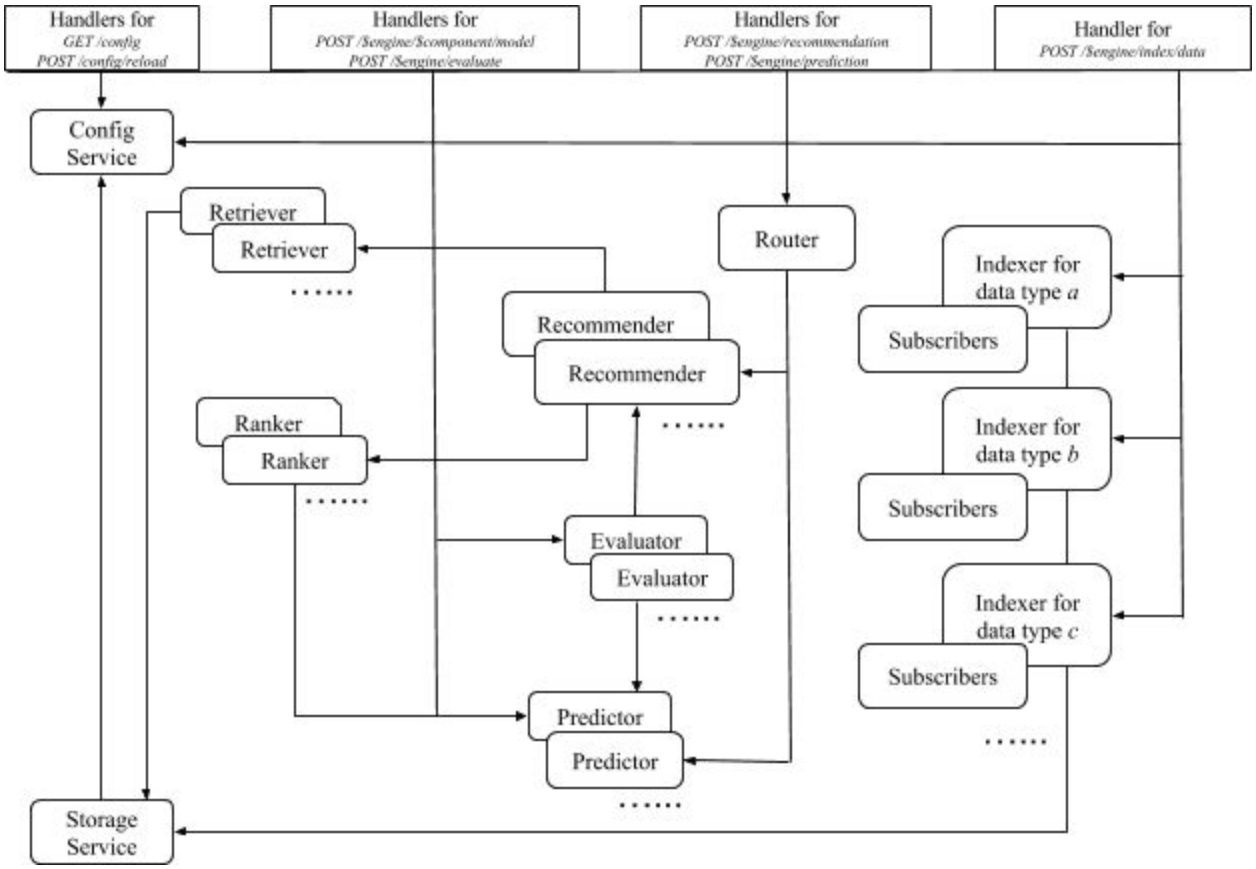

Figure 3. Samantha Data/Request Processing Flows. The directions of the arrows represent data flow or component dependencies.

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. As illustrated in Figure 3 below, when Samantha receives configuration related requests, it directly delivers it to the global Config Service.

2. model management

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

When model management requests come, Samantha first tells which type of component it is, e.g. Retriever and then finds the right retriever (recall that there could be multiple components of the same type differentiated with names) by reading the name from the request. After this, the identified component is responsible for doing the actual work of managing its models or evaluating a predictor/recommender with the given input data based on the request (with arguments/parameters).

3. recommendation and prediction serving

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

If users are requesting recommendations or predictions, Samantha first asks Router to identify a recommender or predictor (Depending on implementation, Router itself consults Config Service to run routing algorithms). Then the identified recommender or predictor is responsible for generating necessary results. Before returning results to users, a common wrapping process writes relevant information on the working recommender or predictor into the response in order to let users know who generates the results.

4. data indexing

POST /$engine/index/data

If data are sent in for indexing, Samantha finds the right indexer based on the indicated name in the request and ask it to index data into backend storage system. Normal indexers all support subscribers which means users can tell Samantha to send those data to other components through configuring the configuration file so that other components can update themselves in real-time, e.g. updating a machine learning model according to an online optimization algorithm.

Dive Into Samantha (*With a walking-through demo for comparison*)

Feature Extraction Framework

	Feature Extraction Framework
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	Provide feature extractors but no framework
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	Feature/representation learning
sparkML (mahout)	Provide feature extractors but no framework
prediction.io	Request processing flow
Samantha	Yes

- ***Case 1***: set up a logistic regression using as many information as it's available in the data (or useful data that we potentially can gather, the so-called big LR; in statistical terms, modeling all possible effects in the data to make better predictions)
    - data

userId movieIdrating popularity avgRating releaseYear genres actors runtime mpaa languages tstamp wishlist highRated
xxxx 78 1 1272 3.174921995 2001 Drama,Thriller Jack Nicholson,David Morse,Anjelica Huston,Robin Wright,Piper Laurie,Richard Bradford,Priscilla Barnes,David Baerwald,Robbie Robertson,John Savage,Kari Wuhrer,Jennifer Leigh Warren,Kellita Smith,Bobby Cooper,Jeff Morris,Buddy Anderson,Edward L. Katz,Joe Viterelli,Eileen Ryan,Ryo Ishibashi,Dennis

Fanning,Lisa Crawford,Jay Koiwai,Elizabeth Gilliam,Michael Ryan,Matthew Ryan,Leo Penn,Penelope Allen,Richard C. Sarafian 111 R English 1427176429 32509,117574 0

.....

- scikit-learn as an example

```
>>> measurements = [
...     {'city': 'Dubai', 'temperature': 33.},
...     {'city': 'London', 'temperature': 12.},
...     {'city': 'San Fransisco', 'temperature': 18.},
... ]

>>> from sklearn.feature_extraction import DictVectorizer
>>> vec = DictVectorizer()

>>> X = vec.fit_transform(measurements).toarray()
array([[  1.,   0.,   0.,  33.],
       [  0.,   1.,   0.,  12.],
       [  0.,   0.,   1.,  18.]])

>>> vec.get_feature_names()
['city=Dubai', 'city=London', 'city=San Fransisco', 'temperature']

>>>lr = LogisticRegression()

>>>lr.fit(X, Y)
```

- Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-biglr-predictor*

Generalized Matrix Factorization

- SVDFeature or Factorization Machine (a working horse)

	Generalized Matrix Factorization
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	Yes
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	difactor: Yes

dmlc: mxnet theano caffee tensorflow	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

- ***Case 2***: use all features from previous LR in a generalized matrix factorization or latent factor/low-dimensional embedding model; modeling almost all possible first order and second-order interaction effects in the data
    - APEX SVDFeature as an example

line:= r k n m <global features> <user features> <item features>
r := prediction target
k := number of nonzero global features
n := number of nonzero user features
m := number of nonzero item features
<global features> := gid[1]:gvalue[1] ... gid[k]:gvalue[k]
<user features> := uid[1]:uvalue[1] ... uid[n]:uvalue[n]
<item features> := iid[1]:ivalue[1] ... iid[m]:ivalue[m]

    - Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-svdfea-predictor*

- ***Case 3:*** implicit feedback, incorporating wishlisting (non-consumptive) data to predict normal rating data
    - APEX SVDFeature: similar as above
    - Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-svdfea-wishlist-predictor*

{
extractorConfigClass = "org.grouplens.samantha.server.featurizer.SeparatedStringExtractorConfig"
attrName = "wishlist"
indexName = "FACTORS"
feaName = "wishlistFact"
separator = ","
}

- ***Case 4:*** buddy recommender

userId, movieId, rating, tstamp, buddyUserIds (separated with comma), avgRating (response/label/objective/feedback)

Indexers for Data In/Out

- ***Case 5***: real-time data and offline data indexing, dumping previous data sets into Samantha to be maintained and reused
    - Samantha: *conf/movielens-demo.conf -> indexer: movieData, userMovieRating, userMovieRatingHistory*
- ***Case 6***: grouped data points output, learning to rank and (grouped) recommendation evaluation
    - Samantha: *conf/movielens-demo.conf -> indexer: groupedUserMovieRating*

	Real-time data indexing and reusing
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	Yes (based on HDFS etc.)
Samantha	Yes

Evaluator and Metrics

	Evaluator and Metrics
Lenskit	Yes
MyMediaLite	Yes

SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	Yes
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	Yes
prediction.io	No
Samantha	Yes

- indicate the predictor/recommender and evaluation data dao
- Samantha: *conf/movielens-demo.conf -> indexer: RealPredictionEvaluator, BinaryClassificationEvaluator, RecommendationEvaluator*

Learning Data, Learning Model and Optimization Solver

- ***Case 7***: oracle-based optimization framework
    - parametric model
    - loss function: normal pointwise loss and listwise ranking loss
        - linear regression
        - logistic regression

Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-svdfea-map-predictor*

```
objectiveConfig {
  objectiveClass = "org.grouplens.samantha.server.objective.MAPLossConfig"
  N = 24
  sigma = 1.0
  threshold = 4.0
}
```

    - (online) optimization method using stochastic oracles
        - sgd and mini-batch
        - async parallel
        - proximal gradient

	General Learning Data/Model and Solver
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	Yes
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	Yes
sparkML (mahout)	Yes
prediction.io	No
Samantha	Yes

Model Serving and Management

- ***Case 8***: continuous maintenance of the models
    - Samantha: *conf/movielens-demo.conf -> indexer: useMovieRating*

```
dataSubscribers = [
  {
   name = "ml-demo-linearucb-predictor"
   componentType = "PREDICTOR"
   requestContext {
    modelOperation: "UPDATE",
    modelName: "ml-demo-linearucb-predictor-model",
    predictor: "ml-demo-linearucb-predictor"
   }
  }
 ]
```

	Model Serving and Management
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	Yes
Samantha	Yes

Flexible Component and Model Dependency

- Component Construction
  - system initiation phase
  - request serving phase
- recommender depending on another recommender/reinforcement learning (see "*conf/movielens-spirit.conf, predictor: sp-ra-build*")
- ***Case 9***: model ensemble/stacking/boosting/composition
  - Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-xgboost-predictor, ml-demo-xgboost-ensemble-predictor*

```
{
    extractorConfigClass = "org.grouplens.samantha.server.featurizer.SVDFeatureFactorExtractorConfig"
    indexName = "TREE"
    predictorName = "ml-demo-svdfea-predictor"
    modelName = "ml-demo-svdfea-predictor-model"
```

```
        feature2dependents {
         "userId" = ["userIdFact"]
         "movieId" = ["movieIdFact"]
        }
}
```

	Flexible Model Dependency
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	Yes
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Retriever for Candidate Generation

- ***Case 10***: tackle with model complexity and large item space
    - Samantha: *conf/movielens-demo.conf -> retriever: FeatureSupportMovieRetriever, RedisMovieRetriever*

	Retriever Component of Recommender
Lenskit	No
MyMediaLite	No

SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Data Expanding and Filtering

- ***Case 11***: fill in data in online/production environment
    - Samantha: *conf/movielens-demo.conf -> indexer: userMovieRating*

```
expandersConfig = [
  {
   expanderClass = "org.grouplens.samantha.server.expander.RedisBasedJoinExpander"
   expandFields = [
    {
     prefix = "movieData"
     keys = ["movieId"]
     fields = ["movieId", "releaseYear", "popularity", "avgRating", "genres",
       "actors", "runtime", "mpaa", "languages"]
    }
   ]
  }
]
```

	Data Expanding or Stream Processing
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	Yes
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	Yes
prediction.io	Yes
Samantha	Yes

A/B or Bucket Testing / Between/Within-Subjects Experiment / Evaluation Loop

- ***Case 12***: Experiment Group Assignment/Tagging
    - Basic router and a hash router

```
router {
  configClass = "org.grouplens.samantha.server.router.BasicRouterConfig"
  recommenderKey = "recommender"
  predictorKey = "predictor"
 }

 router {
  configClass = "org.grouplens.samantha.server.router.HashBucketRouterConfig"
  predictorConfig {
    hashAttrs = ["userId"]
    numBuckets = 100
    name2range {
```

```
    "ml-demo-svdfea-predictor": [10, 99]
    "ml-demo-xgboost-predictor": [0, 9]
   }
  }
  recommenderConfig {
   hashAttrs = ["userId"]
   numBuckets = 100
   name2range {
    "ml-demo-svdfea-recommender": [10, 99]
    "ml-demo-xgboost-recommender": [0, 9]
   }
  }
 }
```

- Samantha recommendation/prediction request response: recommendations are tagged/marked with the corresponding recommender/predictor


```
'{
"status":"success",
"data":{
        "recommendations":{"limit":24,"offset":0,"maxHits":2750,"ranking":[{"score":2.846588150057
        584,"attributes":{"support":396.0,"movieId":"162376","userId":296986},"instance":"rO0ABXNy
        ADxvcmcuZ3JvdXBsZW5zLnNhbWFudGhhLm1vZGVsZXIuc3ZkZmVhdHVyZS5TVkRGZWF0dXJ
        lSW5zdGFuY2UAAAAAAAAAAQIABUQABWxhYmVsRAAGd2VpZ2h0TAAFZ2ZlYXN0ABBMam
        F2YS91dGlsL0xpc3Q7TAAFaWZlYXNxAH4AAUwABXVmZWFzcQB+AAF4cgBCb3JnLmdyb3V
        wbGVucy5zYW1hbnRoYS5tb2RlbGVyLmZlYXR1cml6ZXIuQWJzdHJhY3RMZWFybmluZ0luc3Rh
        bmNlHJRin5RSUkgCAAFMAAVncm91cHQAEkxqYXZhL2xhbmcvU3RyaW5nO3hwcAAAAAAAA
        AAAP/AAAAAAAABzcgATamF2YS51dGlsLkFycmF5TGlzdHiB0h2Zx2GdAwABSQAEc2l6ZXhwA
        AAAA3cEAAAAA3NyADFvcmcuZ3JvdXBsZW5zLnNhbWFudGhhLm1vZGVsZXIuZmVhdHVyaXp
        lci5GZWF0dXJlqBau7Pp4F/YCAAJJAAVpbmRleEQABXZhbHVleHAAAAAAP/AAAAAAAABzcQ
        B+AAcAAAABP/AAAAAAAABzcQB+AAcAAAC9P/AAAAAAAAB4c3EAfgAFAAAAAXcEAAAAA
        XNxAH4ABwAAALo/8AAAAAAAAHhzcQB+AAUAAAABdwQAAAABc3EAfgAHAAAAAD/wAAA
        AAAAAeA=="}, ...]
        },
        "configuration":{
                "retriever":"FeatureSupportMovieRetriever",
                "name":"ml-demo-svdfea-recommender",
                "configClass":"org.grouplens.samantha.server.recommender.StandardRecommenderCon
        fig",
                "ranker":"ml-demo-svdfea-ranker"
                },
        "engine":"movielens-demo"
        }
```

}'

	Experimental Framework
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Online Learning Loop

- group tagging
- ***Case 13***: from data to (feature extracted) instance
    - similarly as the above recommendation/prediction request response: extracted feature representation of the data can be optionally returned to the app

	Online Learning and Evaluation Loops
Lenskit	No
MyMediaLite	No
SLIMx	No

libsvm libFM apex svdfeature	No
scikit-learn (weka)	No
dmlc: xgboost difacto	No
dmlc: mxnet theano caffee tensorflow	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Reinforcement/Bandits Learning

- ***Case 14***: Linear UCB in a system
    - Samantha: *conf/movielens-demo.conf -> predictor: ml-demo-linearucb-predictor*
    - a model stacking example

	Reinforcement/Bandits Learning
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm libFM apex svdfeature	No
scikit-learn (weka)	No
dmlc: xgboost difacto	No

dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Scheduler for Offline Jobs

- ***Case 15***: model retraining (dumping and backup)

```
{
    indexerData = [{
      indexerName = "userMovieRating"
      daoConfigKey = "learningDaoConfig"
      requestContext {
        beginTime = "now - 365 DAYS"
        endTime = "now - 1 DAYS"
      }
    }, {
      indexerName = "userMovieRating"
      daoConfigKey = "validationDaoConfig"
      requestContext {
        beginTime = "now - 1 DAYS"
        endTime = "now - 0 HOURS"
      }
    }, {
      indexerName = "userMovieRating"
      daoConfigKey = "evaluatorDaoConfig"
      requestContext {
        beginTime = "now - 1 DAYS"
        endTime = "now - 0 HOURS"
      }
    }]
    otherData = []
    runner {
      name = "ml-demo-svdfea-predictor"
      componentType = "PREDICTOR"
      requestContext {
        modelName = "ml-demo-svdfea-predictor-model"
```

```
            modelOperation = "BUILD"
        }
    }
}
```

	Embedded Schedulers
Lenskit	No
MyMediaLite	No
SLIMx	No
libsvm **libFM** **apex svdfeature**	No
scikit-learn **(weka)**	No
dmlc: **xgboost** **difacto**	No
dmlc: mxnet **theano** **caffee** **tensorflow**	No
sparkML (mahout)	No
prediction.io	No
Samantha	Yes

Extensibility

Principles

- core Samantha codes are very short and simple. All functionalities are provided by implementing component interfaces which makes it highly extensible.
    - Distributed Protocol Design and Lightweight Core
    - Shallow Dependencies/Flattened Software Layers
- extended interface through request
- separation of config class and actual working class

Implementing New Model/System

- learning to rank (done)
- ***Case 16***: SLIMx
    - Naturally fit in the learning model and optimization solver framework, although handling the constraints is tricky because it needs a new constrained optimization solver (if we insist on the constraints in practice)

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \text{diag}(W) = 0, \end{aligned}$$

- deep learning
    - model composition and chaining

Implementing New Algorithm/Solver

- proximal gradient methods (done)
- regularized dual averaging
- accelerated gradient method
- ***Case 17***: quasi Newton method
    - approximating the second order derivative with the first order ones

Integrating New Model/System

- Case 18: xgboost (done)
- lenskit item-based knn
- Case 19: wikibrain

Scalability

- ***Case 20***: go/extend to cluster mode
    - serving and learning
    - data and model

Index/Variable Space (Parameter Server)

- Samantha defines two key interfaces for extending models and algorithms to distributed systems: IndexSpace and VariableSpace. VariableSpace abstracts the management of scalar and vector variables. Models using VariableSpace request space from it whenever it needs variables. IndexSpace defines the the capabilities of converting any string to an index pointing to a position in the VariableSpace.
- What this means is that if we implement a model using the interfaces of IndexSpace and VariableSpace, we can easily port the model and its learning algorithms to a distributed cluster by

simply switching the implementation of these interfaces, when the size of the model or time complexity of the algorithm requires a distributed system.
- Samantha currently supports two types of spaces. One is based on memory and another one is based on Redis.

Extensible DAO

- All models and algorithms in Samantha use a general DAO interface to read in data. This enables Samantha to recognize different formats or sources of data by extending the DAO implementations. For example, one can easily implement a DAO that reads in data from HDFS (note that writing to different storage destinations are handled by Indexer component)

Redis and Elasticsearch

- Depending on the type of components, users can choose to store and access data from Redis and Elasticsearch. More storage systems will be supported in the future but they are the two most commonly used ones.
- One important case is expanding/joining data before feeding them into a machine learning model to conduct feature extraction, e.g., joining activity data with user and item meta data etc.

Integrating with Applications/Servers

- main integration threads
    - prediction/recommendation request
    - data indexing request
- once-for-all bootstrap

Summary and Outlook

- Production Ready with One Configuration File
- what Samantha provides
- what's the roadmap

Acknowledgement

- thanks to the support of GroupLens and Prof. Joseph Konstan.
- thanks to Max Harper for his feedback on multiple components of Samantha and his teaching in software development especially Java and play framework.

Appendix

More about Installation/Setup

- Intellij remote debugging
- iPython notebook and python requests package
- install extensions
    - xgboost installation
- install redis
- install elasticsearch

Architectures

- real system cases

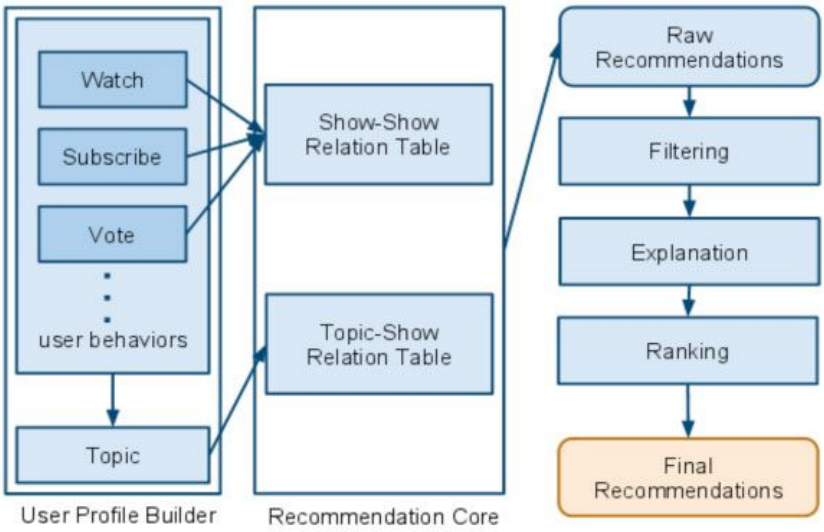

Figure 2. Hulu RecSys Online Architecture[2]

[2] http://tech.hulu.com/blog/2011/09/19/recommendation-system.html

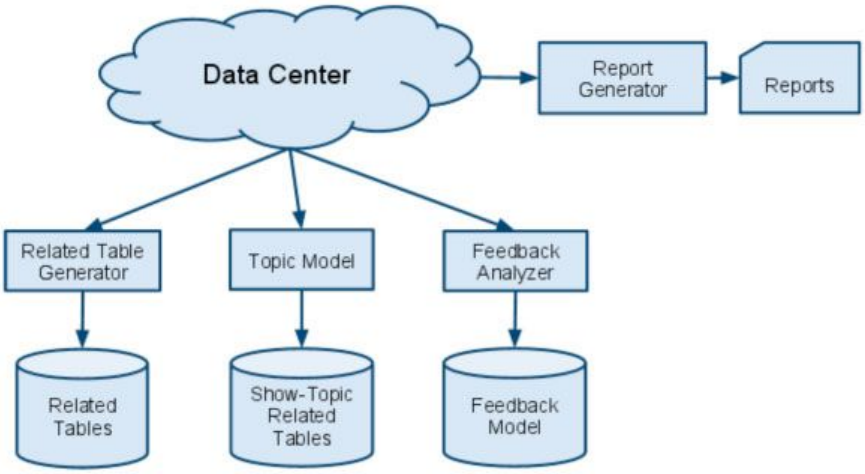

Figure 4. Hulu RecSys Offline Architecture

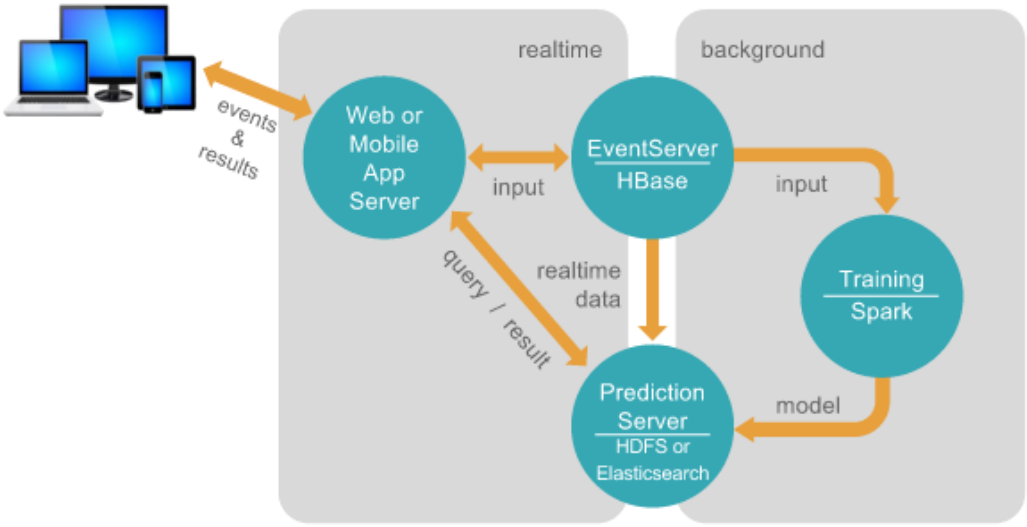

Figure 5. Prediction.io architecture[3]

[3] http://predictionio.incubator.apache.org/system/

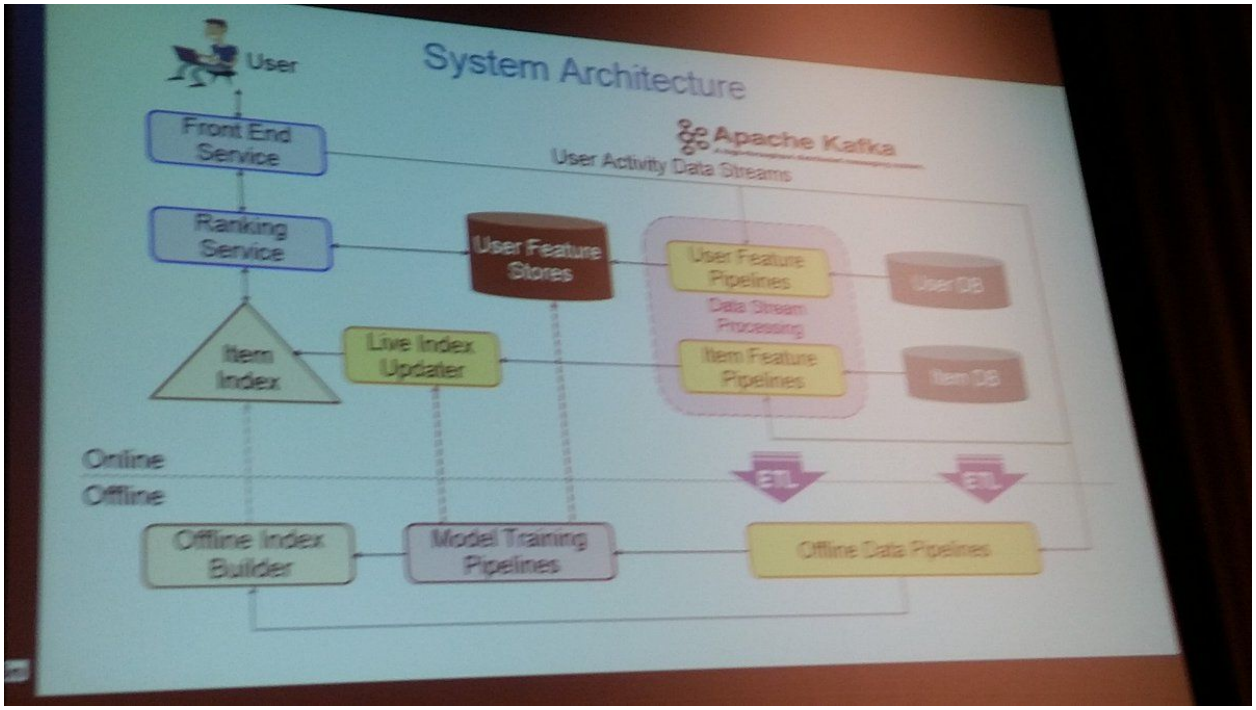

Figure 6. LinkedIn RecSys Architecture.

- Samantha architecture (See the graph of components and request processing flow for another level of details)

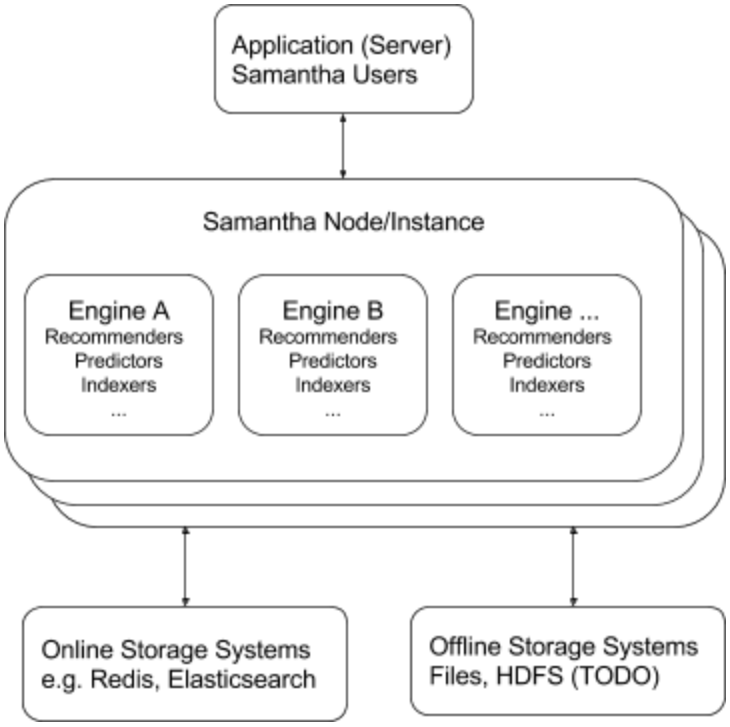

Figure 7. Samantha Architecture

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

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 separation
    - 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

- models
    - linear/logistic regression
    - funk-svd
    - svdfeature
    - factorization machine
    - regression trees
    - gbdt (xgboost)
    - gb-cent
    - bandits learning: linear ucb
    - reinforcement learning: Q-learning
    - feature-based knn
- loss/objective functions
    - regression
    - binary classification
    - ranking
- algorithm
    - stochastic gradient descent
    - proximal gradient method