Rafiki: ML as an Analytics Service System

Authored by Wang et al.

Presented by Varshanth R Rao

Agenda

1) Why Rafiki?
2) Model Training
   - Model Selection
   - Distributed Hyperparameter Tuning
3) Model Inference
4) Experiments & Analysis
5) Conclusions
6) Discussion

1. WHY RAFIKI?
The Rise of Advanced Analytics

Big Data Sources

- Product/Service Reviews
- Device Generated Content
- Uploaded media: Images/Videos

Complex Analytics

- Sentiment Analysis
- Content Filtering
- Image classification/Object Detection/Image Segmentation/Video Analysis

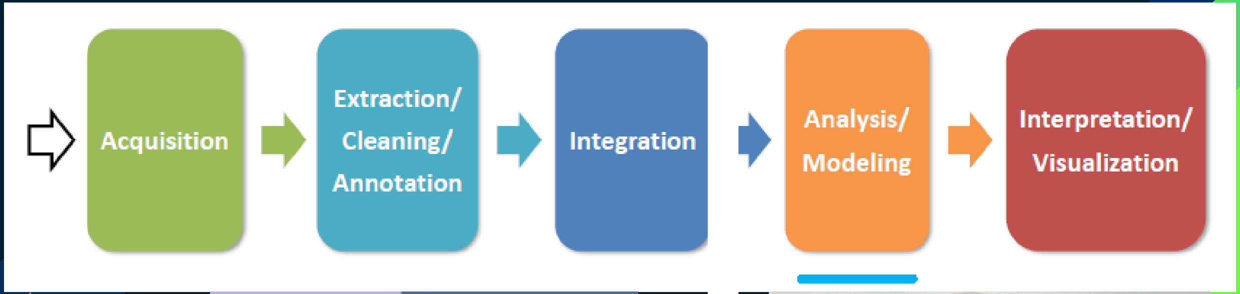
1. WHY RAFIKI?
The Problem

1) Expertise Knowledge Required to train ML algorithms to data and integrate with UDFs

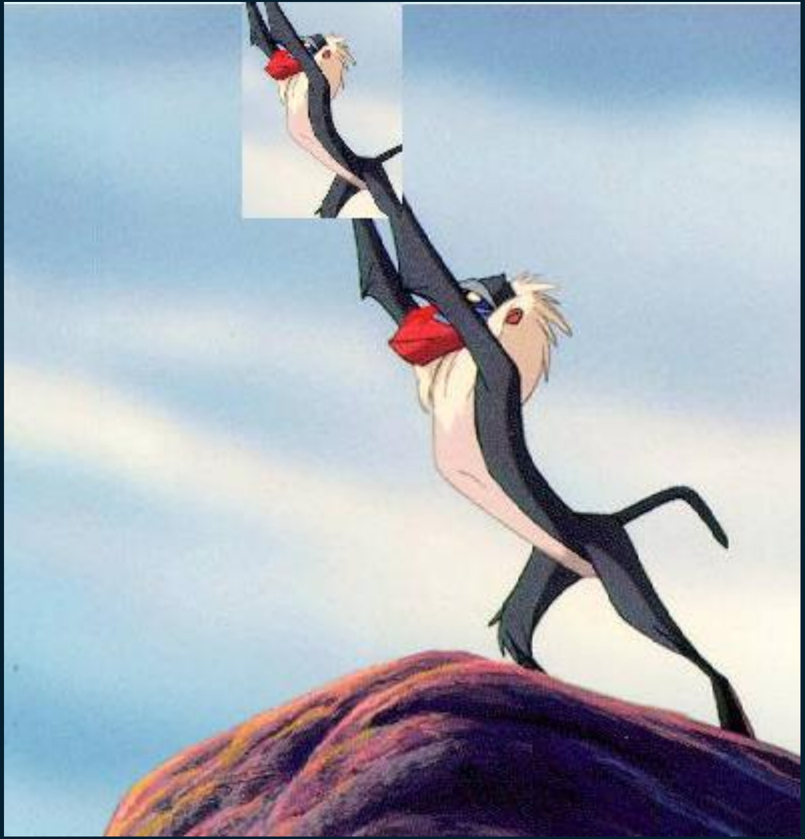
2) Use of external cloud services (AWS, Azure, GCP) hinders flexibility to use own data (for training) or own customized model for solving problems

3) Numerous knobs: Hyperparameters (Number of layers, learning rate etc.)

1. WHY RAFIKI?
Non ML Users be like..



1. WHY RAFIKI?
Enter Rafiki....



1. WHY RAFIKI?
Enter Rafiki….

1) Dataset
2) Training Config

Model Selection

Model Training

Model Inference

Resources

Rafiki

Model Deployment

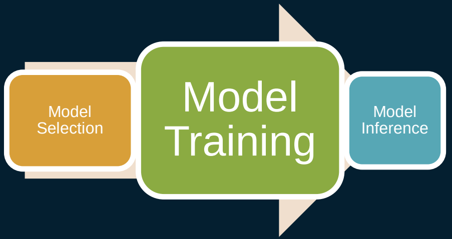
Rafiki Overview:

- *Users configure training/inference jobs through RESTFul API/SDK*
- *For each task, Rafiki provides built-in models (ML Framework agnostic)*
- *Users can monitor the training job*
- *Parameters of the trained model are stored in distributed systems*
- *Users deploy the trained model*

2. Model Selection → Model Training → Model Inference

Task	Models
Image classification	VGG, ResNet, Squeezenet, XceptionNet, InceptionNet
Object detection	YOLO, SSD, FasterRCNN
Sentiment analysis	TemporalCNN, FastText, CharacterRNN
...	...

2.



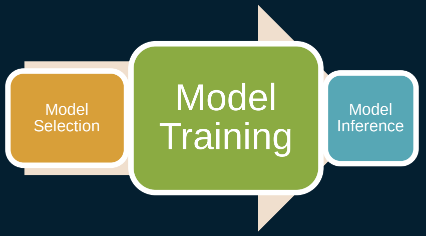
Hyperparameter Tuning:

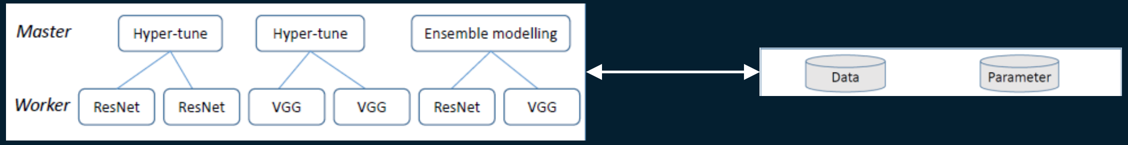
Group	Hyper-parameter	Example Domain
1. Data preprocessing	Image rotation Image cropping Whitening	[0,30) [0,32] {PCA, ZCA}
2. Model architecture	Number of layers N_cluster Kernel	Z^+ Z^+ {Linear, RBF, Poly}
3. Training algorithm	Learning rate Weight decay Momentum	R^+ R^+ R^+

```
class HyperSpace():
  def add_range_knob(name, dtype, min, max,
          depends=None, pre_hook=None, post_hook=None)

  def add_categorical_knob(name, dtype, list,
          depends=None, pre_hook=None, post_hook=None)
```

2.





Distributed Tuning

1) Master iterates over hyperspace and distributes trial to workers

2) Worker trains model with passed hyperspace & reports back to master

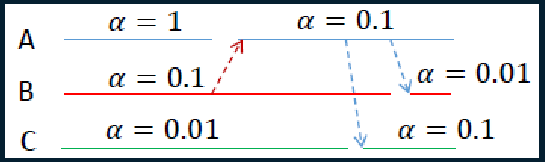
3) Trial advisor on master generates next trial

4) Master stops when there no more trials or stopping criteria is satisfied

5) Best parameters stored in the parameter server

Collaborative Tuning

- Uses concept of pretraining to initialize new trials with parameters of existing well performing trials from other workers



- Also activated by α-greedy strategy to solve the problem of bad parameter initialization

3. Model Selection → Model Training → Model Inference

S = Request List

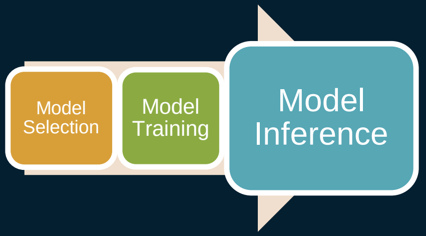
τ = Latency Requirement

l(s) = latency of single inference

$$\min \frac{\sum_{s \in S} max(0, l(s) - \tau)}{|S|}$$

- Larger architectures/Ensembles -> Better Accuracy
  -> Larger Latency
- Goal: Take advantage of Parallelism using GPUs using larger batch size for inference
- Overall idea is to allow l(s) to be at max τ in an effort to maximize the batch size

3.



Single Inference Model

1) Read requests from request queue

2) If the number of requests in queue is larger than max batch size Bmax, then service Bmax requests (older first)

3) If the sum of the time required to perform inference on current batch and the waiting time to fill the next best max batch size is greater than τ, service the current request queue

Multiple Inference Model

- Assigns a reward function for prediction accuracy while penalizing overdue requests

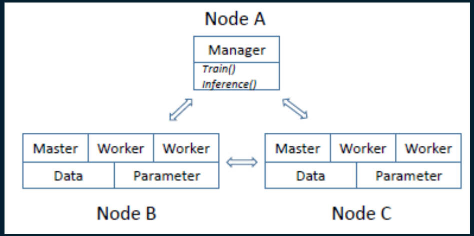
$$\max R(S) - \beta R(\{s \in S, l(s) > \tau\})$$

- State: feature vector representing inference time of each model & waiting time of all requests in queue

- Action: decide batch size & model selection

$$a(M[\mathrm{v}]) * (b - \beta|\{s \in \mathrm{batch}|l(s) > \tau\}|)$$

4. Experiments & Evaluation



Deployment

- Kubernetes managed docker containers

- Dockers represent new models, hyperparameter tuning algos, ensemble methods, application code & libraries

Storage & Distribution

- Data nodes using HDFS stores datasets

- Parameter server with caching used for storing models

- Nodes (dockers) of the same job are located on the same machine to avoid network communication

Experimental Setup

- 3 machine topology

- NVIDIA 1080Ti GPU

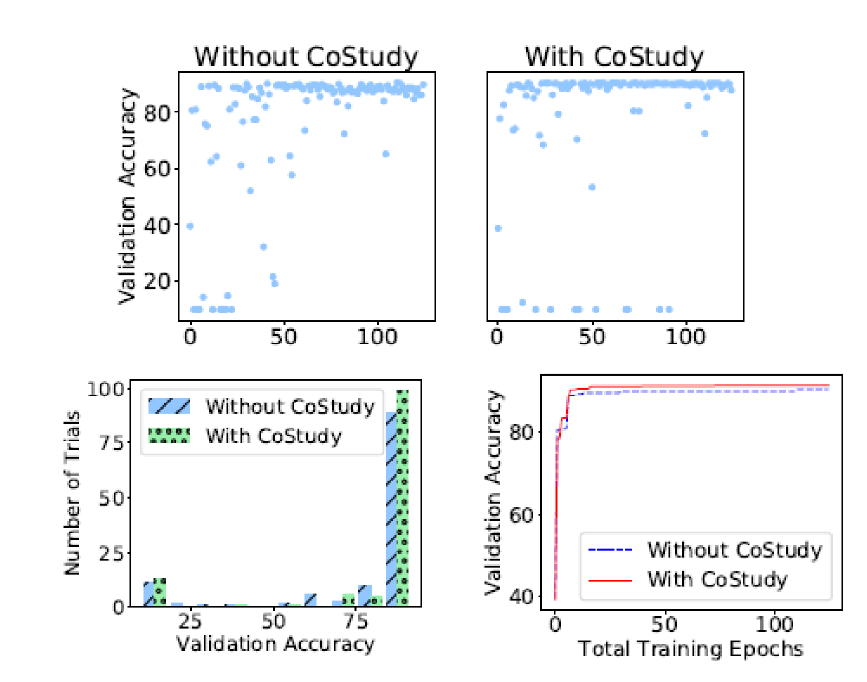
- 64 GB RAM

- Training Study: CIFAR10 Dataset

- Inference Study: ImageNet Dataset

4. Experiments & Evaluation: Hyperparameter Tuning

- CoStudy yields better accuracy
- CoStudy conducts more trials at higher accuracy levels i.e. does not waste trials on low accuracy hyperparameters
- Bayesian Optimization is a better TrialAdvisor
- Execution time decreased as number of workers increased



4. Experiments & Evaluation: Inference

Setup

- Model the service request policy using a sine function
- Modulates between high (dense) and low (sparse) service request densities

Single Inference Model

- RL algorithm performs similar to greedy when rate is high and better when rate is low (RL services overdue slow filling queues)

Multiple Inference Models

- Greedy algorithm accuracy remains constant / within consistent band
- RL algorithm accuracy in the same range as Greedy when rate is high but higher when the arrival rate is low
- Overdue requests significantly lesser using RL

5. Conclusions

Why Rafiki?

Decouples DB tasks from analytics complexities

Handles training & inference services so users can concentrate on application logic

Model Training

Model selection: Tasks -> Algo Map

Distributed Hyperparameter Tuning

Collaborative Tuning

Model Inference

Use of batch size to parallelize inference

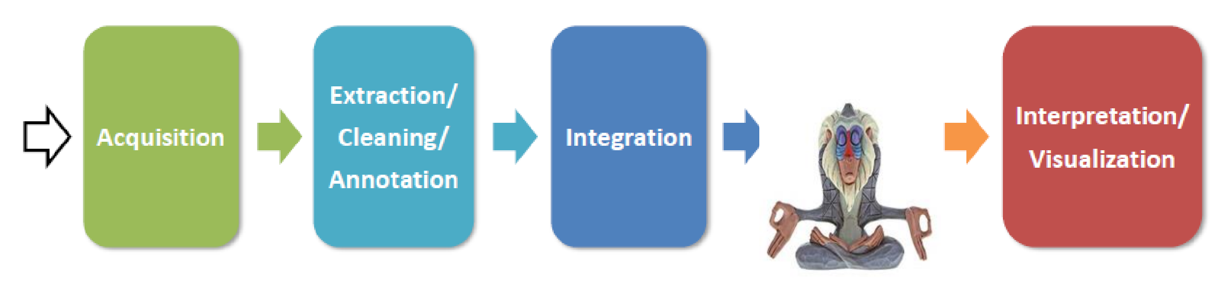
Latency-Accuracy Tradeoff

Multiple Inference Models: Request Driven Model Selection

Key Takeaways:

Rafiki provides a framework agnostic abstraction to use ML/DL algos in applications without having to worry about the burdens of algorithm choice and training difficulties

Rafiki provides requirement driven model selection & distributed hyperspace searching capability to extract the most from the models



5. Discussion

Light Note: Why do they name the system Rafiki??

Paper Specific

1) Authors pointed out lack of ability of using own model with external cloud services but they also do not provide the ability to use customized models

2) Training & inference jobs are distributed across nodes but a single job (training/inference task) is still on 1 machine -> Not using multiple GPUs or multiple machines to take advantage of H/W resources

3) Why do the authors train on such a small dataset (CIFAR10) while inferencing on a large dataset (ImageNet)? What about other complicated tasks like object detection, sentiment analysis etc.? Experimentation seems inadequate.

Looking Ahead: AaaS

1) In-memory models to service inference requests: Challenges (Model complexity, Limited GPU Memory, etc.)

2) 2 Different directions: Mobile/Integrated AI Chips vs On Cloud AaaS

THANKS!