Project: Distributed Hyperparameter Tuning System

Akshitaa Jay, Ryan Mechery, Aditi Ravindra, Aryan Sajith

Project Overview

The goal of this project is to design and implement a distributed system that performs hyperparameter tuning across multiple machines. The goal is to design a scalable workflow that can manage and execute multiple parameter search tasks in parallel. We plan to use Kafka to coordinate the distribution of tasks and simulate multiple worker nodes through Docker containers. We aim to obtain a better understanding of distributed task management, parallel computation, and performance evaluation in a system-level context through this project.

Milestone Goals

- **Implement Simple Version of Random Search Hyperparameter Tuning Algorithm** - Write a Python implementation of Random Search to form the foundation of the tuning system. This function will take in a user defined range of hyperparameter values. For categorical values we will use `random.choice` (like batch size of {16, 32, 64}) and for uniform values we will use `random.uniform` (like learning rate between 0.001–0.1).
- **Develop model training routines for sci-kit learn** - Develop modular training functions that accept hyperparameters as input and can execute model training for sci-kit learn.
- **Set up a Kafka environment and demonstrate simple communication** - Configure a local Kafka cluster to manage hyperparameter tuning tasks as messages. Verify that the messages can be sent, queued, and received/read successfully across simulated nodes.
- **Extend Kafka workflow to handle distributed task management** - Extend the Kafka workflow to support parallel task assignment and tracking, allowing multiple hyperparameter tuning jobs to be processed simultaneously by different worker nodes.
- **Begin Dockerizing Model Training Experiments** - Begin dockerizing model training component, ensuring containers contain necessary dependencies for running isolated training experiments. Also, ensure that the Kafka workflow can send and receive messages with these Docker containers for testing in the final milestone.

- **Evaluate Learning Rate and Batch Size Hyperparameter Tuning for Initial System Performance** - Develop training experiments using models such as K-Nearest Neighbors (KNN), Logistic Regression, and Decision Trees to test system functionality with learning rate and batch size hyperameters. Record runtime and task completion rate metrics for a preliminary evaluation.

Final Project Goals

- **Finalize Docker components** - Deploy up to 10 containers that simulate worker nodes communicating through Kafka. Each container should be capable of training models independently and streaming results back.
- **Fault-tolerant management** - Implement basic fault tolerance by manually simulating worker failure, could be via exceptions, to validate task re-queue, re-assignment and completion.
- **Conduct Comprehensive Testing** - Complete testing to measure 3 - 5 system performance metrics (such as runtime, throughput, model storage needs) across 3 -5 different ML models (such as logistic regression, decision trees, KNN, and simple MLP) and varying up to 3 auxiliary configurations such as containers, task volumes, and message sizes.
- **Final performance charts & analyses:**
    - **Overall Note:** We will look to perform small-scale informative systems experiments such as limiting workers up to 10, limiting per-worker executions up to 30 minutes, and so forth to ensure time-feasible and informative systems analysis for this project.
    - **Runtime vs. number of containers chart** - Can be useful to analyze if increasing the number of containers positively correlates with decreased runtime and increased performance and up to what threshold (if any) this trend holds and if any performative regression emerges.
    - **Task distribution across workers chart** - Visualizes how balanced and well distributed our hyperparameter tuning system is. This may be measured in terms of tasks split between each worker or time splits of each worker in a distributed execution run.
    - **Throughout & completion efficiency chart** - Lastly, this chart is evaluating the proportion of completed tasks to assigned tasks across varying workload demands to benchmark the overall efficiency and scalability of our system.
- **Final codebase submission** - Ensure all Kafka scripts, hyperparameter training modules, and Dockerfiles are uploaded to GitHub with inline documentation and a README.