Cloud Computing, Distributed Computation, and MLOps – Assignment

MMA Colloquium 2024

Introduction

You are Donny Wrench, the CCO (Chief Chess Officer) of Chess.net, the premier online chess platform. Lately, revenue has been suffering due to those pesky hippies at Lichess.org stealing your users with their dastardly plan of being free, open-source, and stubbornly refusing to show ads. Now you're in the unfortunate position of needing to fundraise to keep the lights on.

Unfortunately, the snobby elite Silicon Valley VCs are immune to your charms, and fail to see the incredible growth potential of a 1500-year old game. They're only interested in flashy AI projects these days, and insist you find a way to incorporate "Machine Learning" into your platform.

You briefly considered using AI to detect and ban cheaters, but quickly dismissed the idea after realizing that revealing the extent of cheating on the platform would be a PR disaster. Instead, you've decided to build a top-of-the-line AI model to predict the outcome of chess games before they even begin. If that doesn't impress Sequoia Capital, nothing will.

Some of your engineers protested against the idea, saying that it's a waste of effort since the Elo and Glicko2 rating systems already establish a baseline probability of winning for each player. But you know that the VCs won't be impressed by a mere "rating system" - they want "Machine Learning". After firing the Luddites, you've decided to go ahead with the plan.

To make your victory even sweeter, you decide to use Lichess' own public database of games to train your model. They foolishly publish a complete dataset of every game played on their site, including the ratings of the players, the time control, and the outcome of the game. You're going to use their own data against them to build the best chess prediction model the world has ever seen (other than the Elo and Glicko2 rating systems, of course).

The project has been broken into three distinct phases below, each one implementing a different phase of the machine learning pipeline. Your task is to complete **at least one** of the three phases, although if you're feeling ambitious you can certainly complete more. Each one is designed to exercise a different layer of the machine learning stack, so you can choose the one that best fits your skills and interests.

Phase 1 – Data Collection

In the data collection phase, we need to gather the game records that will inform the model's expectations about a player's expected performance given their rating. The Lichess database is available here and is in the form of a series of compressed PGN files. What we need to

I recommend testing your code on a small subset of the data, to avoid very long iteration times. If you download the early years (2013-2015) of the database, it should only consume a few gigabytes of disk space while sill containing the same structure as the full dataset.

The goal of this phase is to produce a Spark DataFrame that contains the following columns:

Column Name	**Data Type**	**Description**
`black_rating`	`int`	Elo rating of the Black player
`white_rating`	`int`	Elo rating of the White player
`time_control`	`str`	Time control of the game
`result`	`int`	Outcome of the game (-1, 0, 1)

Table 1: Structure of the Spark DataFrame

The `result` column should be `-1` if Black won, 0 if the game was a draw, and 1 if White won. (Games without a result, for whatever reason, can be omitted.) You may create other columns if desired, if you think they might improve the overall prediction accuracy, but at least the ones above are required.

In order to complete this phase, you will need to write a Spark job that reads the PGN files from the Lichess database, parses the PGN files into a DataFrame, and then writes the DataFrame to a Parquet file. You may use a specialized PGN-parsing library if you choose, but this is largely unnecessary since we only care about the headers of the PGN files, which are simple to parse manually.

The final output should be a serialized DataFrame in any format (ideally Parquet) that can be read by the next phase of the pipeline.

Note: A hint will be given live in lecture that will make this phase much easier to complete.

Phase 2 – Model Training

Now that we've cleverly stolen Lichess' data and stored it in a format that's easy for a training job to consume, we can move on to the next phase: training the model to predict the outcome of a game given the ratings of the players and the time control.

In order to track the lifecycle and performance of the model, we will use MLflow to log the parameters, metrics, and artifacts of the model training process. This will allow us to easily compare different models and track their performance over time. MLflow is beautifully documented and has a very simple API, so you should be able to get up and running with it very quickly using the quickstart guide and Python API documentation.

You will need to write a Python script that reads the Parquet file from the previous phase, and uses it to train a model to predict the outcome of a game. It is up to your judgment which features to use in the model, and which model to use. You simply need to predict the classification of the `result` column.

The emphasis on this phase is not on the quality of the model, but on the process of training the model and logging the results. You may use any model you like, but a simple logistic regression model is probably sufficient for this task. The goal is to demonstrate the process of loading the dataset, training a model, and logging the results, not to produce the most accurate model possible. You may use any machine learning library you like, but you will find the logging *much* easier if you use one of the libraries that has built-in MLflow support.

For the purposes of these last two phases, I have set up a publicly-accessible MLFlow instance that you can view at https://mlflow.apetre.sc/. You will be given a username and password in class that you can use to log in and view the results of your run. Note that when running the quickstart guide above, you do not need to go through the steps of setting up your own MLFlow server, as I have already done this for you. Simply set the `MLFLOW_TRACKING_URI` to `https://mlflow.apetre.sc/` and use the username and password you were given in class.

When logging a run, please use an Experiment name that corresponds to your username on Quercus, so I can easily correlate your runs with your submission.

Note: If you chose to also complete Phase 1, you should use your own dataset that you created in Spark, in whatever format you chose to save it in. If you did not complete Phase 1, you can use a copy of the dataset that I created in Parquet format. My dataset is located in a public S3 bucket at `s3://rotman-cloud-data/mlflow/data`. You can either access it directly from S3, or download it over the HTTP gateway at:

`https://rotman-cloud-data.s3.amazonaws.com/mlflow/data/[filename]`

(If you do implement Phase 1, feel free to use these files as a way to verify that your own dataset matches the expected structure.)

Phase 3 – Model Evaluation

Now that you've trained a model and logged the results, it's time to evaluate the model's performance. The goal of this phase is to load the model from the previous phase, evaluate its performance on a held-out test set, and log the results to MLflow.

We are going to use the mathematical properties of the Glicko2 rating system to create a method for evaluating our model's performance probabilistically. Although our model is only predicting the outcome of a game as a pure classification, we can use the ratings to calculate the expected probability of winning for each player, and so separate our test set into rating bands and calculate the expected win rate for each band. We can then compare our classification error within each band and see if our model's error rate is proportional to the rating difference betwen the players.

Although the actual math behind the Glicko2 system is quite complex, we will simplify it by saying that between two players of equal rating, the probability of each player winning is 50%. For each 100 rating point difference between the players, the expected win rate for the higher-rated player increases by 10%. So, for example, a 2000-rated player is expected to win 70% of the time against a 1800-rated player, and 90% of the time against a 1600-rated player. If the rating difference is greater than 400, we can consider that game unrated and exclude it from our analysis.

So, for this phase, we are going to download the model from the MLflow model registry, load it into our code, and use it to predict the outcome of the test set. We will then calculate the expected error rate for each rating band difference between -400 to +400 and log it in the MLflow tracking server with `mlflow.evaluate` function.

(The more mathematically-minded among you may recognize that it would be more accurate to train our model to predict the expected win rate directly, rather than the outcome of the game. Feel free to do this if you want to challenge yourself, but it is not required for this phase.)

Note: If you chose to complete Phase 2, you should use your own model in your own experiment name for this phase. If you did not complete Phase 2, you should use the model that I trained in my own experiment for this phase. My experiment name is `apetresc`.