Carnegie Mellon University

04800 Special Topics: AIOps, Fall 2024

Lab 3: Detecting anomalies with Prophet

Expected individual study hours: 12

Points: 100

Purpose

The goal of this third lab is to apply your Prometheus and Grafana skills to the first test for anomalies using a simple forecasting framework called Prophet. You'll learn how to capture metric data using the Prometheus API and then monitor your metrics using Prophet to detect any outliers. You'll then re-post those anomalies back into Prometheus and visualize in Grafana as an additional metric. Prophet will be used in later labs to capture the metrics we'll monitor from the Boutique application running in Kubernetes.

Target Learning Objectives

- Understand the theory and mechanisms for time-series forecasting using Prophet.
- Gain a hands-on keyboard familiarity with building Prophet models in Python using the Prometheus API to extract metric data and republish anomalies.
- Visualize anomalies in custom dashboards in Grafana.

Lab Task – Building the Prophet model

1. Revisit the app_one Python code from Lab 2 and confirm you have two new metrics called train_gauge and test_gauge. These should be gauge metrics which simulate a request time but where train_gauge clips values above some threshold such as 600 msec. You should have defined the metric in the last lab as a random value between 0 and 0.6. We'll use this metric to train our Prophet model. The second metric, test_gauge, should be a random variable gauge metric between 0 and 1 and will be used to detect values above the threshold as anomalies using Prophet.

2. Rebuild your app_one with the two new metrics and restart your existing Prometheus sandbox. Visit Prometheus in your web browser and query your metrics to confirm they are running.
   Hint: Remember you can also visit localhost:8000 to get a snapshot of all metrics available.

3. Add your two metrics to your Grafana dashboard as simple time series (graphs over time of a single value) and take screen shots to confirm they are running.

4. Using the sample Notebook from class, prepare a Python application (not Notebook) that reads the training data metric for a period of time, builds a Prophet model from it, and evaluates the model against the test metric. Simply print the anomalies detected to the console log.

5. Update your application to repeat the model test cycle (fetch data, test model) on each 60-second iteration. Continue this loop indefinitely printing results to the console.

   Important hints:

   - Be careful to not overlap your train and test times, e.g. pull the training metric for the last 5 min, wait a minute, then pull the test metric just for the previous minute. Then evaluate the model against the past minute's test metric for anomalies.
   - Be sure to specify "growth='flat' in your model or you will very find that the model drifts rapidly even after just a few minutes and you start getting "unreasonable" anomalies.

Lab Task – Package model as Docker container/image

1. Package your monitoring application as a Docker image and add it to the docker-compose.yml spec so you can run it inside the collection of other docker containers. You'll need to create the Dockerfile, update the docker-compose.yml with your monitor, and also importantly update the Prometheus config file to sweep metrics from your application.

2. Add a gauge metric to your monitor application for anomaly-count and set that on each iteration of your model's evaluation. You'll need to add the Prometheus imports to your application along with the declaration of your new anomaly gauge. Note that it is a gauge since the number of anomalies detected will vary both up and down on each iteration through your testing.

3. Add a visualization for your anomaly metric to your Grafana dashboard and take a screen shot of that along with the previous time series and histograms of your request_time metrics.

Lab Task – Explore model quality

1. Compute the model quality metrics Mean Absolute Error and Mean Absolute Percentage Error (MAE and MAPE) as shown in the provided Jupyter Notebook (see cell 39 in updated Notebook in Slack and Canvas).

2. Construct a dataframe with a row for each test step with columns: current time, number of anomalies detected, MAE, and MAPE). Print this dataframe to the console on each test iteration.

3. Create two new Gauge metrics for these error stats and publish them to Prometheus from your app. Add time series visualizations to your Grafana dashboard for MAE/MAPE in addition to the original train/test and Anomaly metric series – should have a total of 5 time series in your final dashboard.

Lab Task – Answer the following questions

1. What do you estimate to be a “reasonable” baseline of data to use this type of Prophet model in an actual running production system? Would that length of training time pose any operational challenges?

2. Do you think such a Prophet model should be allowed to retrain continuously in a production setting or require some manual review/approval? What could be some pitfalls of allowing a fully automatic operation?

Upload the following documents into your lab3 repo in GitHub:

1. your code
2. screen shots of Grafana with working visualizations shown (train, test, anomaly, MAE, MAPE graphs)
3. console log showing training and forecast time results
4. answers to the above questions

Due date Thursday, Sept 26, at 11:59 PM CAT