Machine Learning Engineering Industry Skills Evaluation Framework

Technical Brief

CodeSignal

Skills Evaluation Lab

Published June 2023

Introduction

With the growing need for organizations to take advantage of the surplus of structured and unstructured data they have at their disposal and the rapid development of machine learning technologies, the demand for engineering talent in this domain is ever-increasing. In fact, the demand for Machine Learning Engineers (MLEs) is projected to grow 13-22% between 2020 to 2030.

Despite such overwhelming interest in MLE talent, hiring MLEs is a turbulent and inefficient process. Many engineering and talent teams struggle to evaluate the technical skills of MLE candidates effectively, often using proxies (i.e., previous experience or pedigree) or poorly-designed evaluations that do not accurately or consistently capture candidates' knowledge and skills.

This paper describes a framework for developing simulation-based evaluations that accurately capture signals about the technical skills of MLE candidates at scale. With the growing popularity of MLE, there is a large influx of candidates switching into the field from related fields (i.e., software engineering), as well as growth in MLE-focused academic programs contributing to an increasing number of new graduates entering the field. As a result, hiring teams may often deal with a large volume of candidates, many of whom might possess relevant technical skills but lack MLE-specific experience or pedigree. Framework-based evaluations will allow engineering and talent teams to greatly scale their hiring processes and make effective hiring decisions while providing a fair and engaging experience.

Generally, MLEs focus on developing software related to machine learning, deep learning, and artificial intelligence. To succeed, MLEs must display a diverse skill set, including:

- Cleaning and manipulating data
- Experimenting with and evaluating various models and algorithms
- Following and incorporating research on machine learning, deep learning, and artificial intelligence

- Solving complex problems in effective and efficient ways
- Understanding and incorporating key business metrics when designing machine learning product pipelines
- Deploying and productionalizing machine learning models.

Generally, experienced MLEs, as well as MLEs who work for larger organizations where there is a division of responsibility between those who build machine learning models and those who productionalize them, are primarily responsible for deploying and productionalizing machine learning models, including model implementation, API design, optimization, ensuring efficient machine learning inference, and performance monitoring. This Framework, developed based on consultation with leading MLE subject matter experts, is designed specifically to assess the knowledge and skills required within this domain.

Framework Specifications

The framework is designed to closely simulate the fundamental knowledge and skills a candidate would be expected to have within Machine Learning Engineering roles focusing on productionalizing machine learning models. The framework can be utilized to create evaluations that span different delivery methods, such as pre-screen assessments or technical interviews while providing objective signals by automatically calculating a final score to represent a candidate's skill level.

Evaluations based on this framework consist of four modules that target a breadth of advanced machine learning engineering topics and coding skills related to machine learning model implementation, API design, and performance instrumentation. The specific content will be common across MLE jobs within a variety of industries to avoid any unfair advantages or disadvantages for different candidate populations. To balance the depth and breadth of content and candidate experience, **the evaluation time for this framework is 90 minutes.** Possible scores range from 200 to 600.

Module 1 – Advanced ML Engineering & MLOps Concepts

This module contains **five scenario-based quiz questions** with an average solve time of **5-10** minutes.

Expected Knowledge

- Understanding of theories behind machine learning algorithms, models, and concepts
- Understanding of tools and processes for optimizing model inference
- Understanding of infrastructure needed to deploy models

Can Include

- Multiple choice and fill-in-the-blank questions to measure breadth and depth of ML-related knowledge, for

example:

- Model selection based on business requirements
- Working with high-dimensional data
- Addressing ML inference bottlenecks
- Model staleness and need for continuous training
- Distribution shift
- Optimizing ML inference for cost

Should Exclude

- Complex calculations that require anything beyond a simple calculator or paper and pencil to solve
- Highly domain specific knowledge in any sub-field of ML (e.g., NLP, Reinforcement Learning, computer vision, etc.)

Module 2 – ML Cost, Loss, & Optimization Metrics

This module contains **five scenario-based quiz questions** with an average solve time of **5** minutes.

Expected Knowledge

- Understanding common cost, loss, and optimization metrics used in machine learning model development

Can Include

- Multiple choice and fill-in-the-blank questions to measure breadth and depth of ML-related knowledge, for example:
- Loss
  - Kullback–Leibler divergence
  - Mean Squared Error (MSE)
  - Cross Entropy
  - Mean Absolute Error (MAE)
  - Root Mean Squared Error (RMSE)
- Metrics
  - Area Under the Curve (AUC)
  - F1 score
  - Recall
  - Accuracy
- Optimization
  - Regularization
  - Resampling and balancing

Should Exclude

- Complex calculations that require anything beyond a simple calculator or paper and pencil to solve
- Highly domain specific knowledge in any sub-field of ML (e.g. NLP, Reinforcement Learning, computer vision, etc.)

Module 3 – Data Processing Pipeline

This module contains **one progressive coding question with two levels** focusing on implementing a data processing pipeline using multiple *csv* files using the standard data exploration libraries. The average time for solving this should be around **20-25 min**.

Expected Knowledge

- Understanding data pipeline, cleaning, and processing
- Implementing functions and algorithms used in data processing
- Proficiency in using pandas and Python OS file handling libraries

Can Include

- Implementing a complete data processing pipeline going through different steps:
  - Data extraction and loading
  - Data rationalization
  - Data cleaning
  - Data transformation
  - Data pre-processing
- Designing reliable, scalable, and highly data preprocessing solutions

Should Exclude

- Highly domain specific knowledge in any sub-field of ML (e.g., NLP, Reinforcement Learning, computer vision, etc.)
- Model training
- Data collection

Module 4 – Model Deployment & Monitoring (API Design & Performance Instrumentation)

This module contains **one progressive coding question with two levels** focusing on model deployment and monitoring model performance. The average time for solving this question should be **20-30** minutes, and candidates are expected to write approximately **20-25** lines of code.

Expected Knowledge

- Knowledge of API design or module/library design
- Serialization
- Monitoring
- Tracing
- Scaling strategies

Can Include

- Designing an API Endpoint that serves predictions
- Parsing a HTTP POST request
- Constructing a HTTP response JSON object
- Creating a Python module/library
- Memory and compute utilization and scalability considerations
- Concurrent processing
- Batch inference vs. stream inference
- Implementing a function to understand model performance
- Identifying bottleneck(s) in an API

Should Exclude

- Vendor specific technologies (e.g., GCP Vertex AI, AWS SageMaker)
- Vendor specific knowledge on app performance monitoring
- Model training

Framework Example Content

Below are example questions for each module of the framework[1]. Similar questions are consistently being developed in accordance with framework specifications and monitored on an ongoing basis to minimize the impact of potential leaks that could result in cheating or

[1] Example questions are for reference only, and examples may not match the exact number of questions outlined for each module in the framework.

plagiarism, ensure the reliability and validity of evaluations, and provide relevant and fair candidate experiences through changing industry standards.

Module 1 – Advanced ML Engineering & MLOps Concepts

You are tasked with creating a binary classification model using a very large dataset. For your initial approach, imagine that you are trying to decide between using either a **Naive Bayes (NB)** or **k-Nearest Neighbors (k-NN)** classifier.

If your primary concern is with a model's **prediction accuracy** to establish a baseline, which of the following are valid reasons for using either Naive Bayes (NB) or k-Nearest Neighbors (k-NN)?

Select all correct options.

- k-NN is susceptible to the curse of dimensionality when there are a large number of features, whereas NB is not
- Zero-frequency problem is a larger concern for NB than k-NN if the dataset contains continuous features
- NB tends to be more accurate that k-NN on larger datasets regardless of the value of k
- Non-parametric nature of NB improves its hyperparameter tuning time compared to k-NN
- NB is faster than k-NN for real-time predictions on larger datasets because it is a linear classifier
- None of the above

Module 2 – ML Cost, Loss, & Optimization Metrics

You apply a regularization penalty to a linear regression model you have developed and realize that some of the coefficients in the model have been zeroed out.

Please select the following regularization penalties that might have been used based on the zeroed out coefficients.

Select all correct options.

- L3 norm
- L0 norm
- L2 norm
- L4 norm
- L1 norm
- None of the above

Module 3 – Data Processing Pipeline

Level 1

You are given a dataset containing the results of a sociological survey about the transportation system of a city. Also, the dataset includes automatically collected data about citizens' average travel distance, travel frequency, and expenses.

Feature description:

1. `id (type: int)` – a citizen's unique identification number;
2. `transportCardId (type: int)` – unique identification number of a citizen's transport card. Can be `None` in case the citizen doesn't use one;
3. `age (type: int)` – a citizen's age. Can be `None` if the citizen declined to specify their age;
4. `averageTravelDistance (type: int)` – automatically collected the average travel distance of a citizen daily. Can be `None` if the citizen asked not to share this information;
5. `travelFrequency (type: float)` – automatically collected the average amount of travel of a citizen weekly. Can be `None` if the citizen asked not to share this information;
6. `travelExpenses (type: float)` – automatically collected average travel expenses of a citizen monthly. Can be `None` if the citizen asked not to share this information;
7. `reason (type: str)` – the main reason to use the transportation system for a citizen. Options are `job`, `university`, `fun activities`, `countryside`, `hospital`. Can be `None` if the citizen declines to specify their reason for travel;
8. `satisfaction (type: int)` – a citizen's overall satisfaction with the city's transportation system on a scale from `1` to `10`, where `1` corresponds to `"absolutely dissatisfied"` and `10` corresponds to `"absolutely satisfied"`. **Cannot** be `None`, as the estimate was specifically required to participate in the survey.

Your task is to create a preprocessing pipeline that fills in missing values to prepare this dataset to be used to train a clusterization model. To achieve that, create a preprocessor that includes the following data transformations:

1. Remove the `transportCarId` feature from the dataset.
2. Replace missing values in the `age` feature with a median value of all citizen's ages.
3. For the features `averageTravelDistance`, `travelFrequency`, and `travelExpenses` replace missing values with the average of the values of the corresponding feature.

Notes. To pass the tests, the preprocessor:

1. Should be serialized to a file named `preprocessor.pkl` using `pickle` module.
2. Should support the `fit_transform` and `get_feature_names_out` methods (see `setUpClass` method in `preprocess_pipeline_tests.py`).
3. If you need to write any additional classes needed for deserialization, please put them in the `custom_transformers.py` file. They will be imported automatically.
4. Can provide the output with any order of features.

Level 2

The dataset was extended by three new categorical features:

1. `gender (type: string)` — a citizen's gender. Options are `male`, `female`, and `didn't share` (applied if the person declined to specify their gender). Can be `None` in case information is missing;
2. `favouriteTransport (type: string)` — a model of transport which a citizen named as their preferred or favorite. Options are `tram`, `trolleybus`, `bus`, `underground`, `bicycle`, `no preference`. Can be `None` in case information is missing;
3. `reason (type: string)` — main reason to use the transportation system for a citizen. Options are `job`, `university`, `fun activities`, `countryside`, `hospital`. Can be `None` if the citizen declined to specify their reason for travel;

Your task is to create a preprocessing model that fills in missing values and encodes categorical data using OneHotEncoding in addition the previous level's preprocessing. To accomplish your task, add the following data transformations to the model:

1. Replace missing values in the `gender` feature with "`didn't share`".
2. Transform `gender` feature with `OneHotEncoding`. As a reminder, options are: "`male`", "`female`", "`didn't share`".
3. Replace missing values in the `favouriteTransport` feature with "`no preference`".
4. Transform `favouriteTransport` feature with `OneHotEncoding`. As a reminder, the options are "`tram`", "`trolleybus`", "`bus`", "`underground`", "`bicycle`", "`no preference`".
5. Replace missing values in the `reason` feature using a `bfill` method. As a reminder, `bfill` method is filling the missing values with the next valid value. If there is no valid value after the missing one, replace it with the previous valid value instead.
6. Transform `reason` feature with `OneHotEncoding`. As a reminder, the options are: "`job`", "`university`", "`fun activities`", "`countryside`", "`hospital`".

For the second and fourth steps, new features should be called in format `<original_feature_name>_<option_name>`. For example, a feature corresponding to the "`male`" option of the `gender` feature will be called `gender_male`. Thus, you will have the following list of features instead of `gender`, `favouriteTransport`, and `reason`:

- `gender_didn't share`,
- `gender_female`,
- `gender_male`,
- `favouriteTransport_bicycle`,
- `favouriteTransport_bus`,
- `favouriteTransport_no preference`,
- `favouriteTransport_tram`,
- `favouriteTransport_trolleybus`,

- `favouriteTransport_underground`,
- `reason_job`,
- `reason_university`,
- `reason_fun activities`,
- `reason_countryside`,
- `reason_hospital`.

Notes. To pass the tests, the preprocessor:

1. Should be serialized to a file named `preprocessor.pkl` using `pickle` module.
2. Should support the `fit_transform` and `get_feature_names_out` methods (see `setUpClass` method in `preprocess_pipeline_tests.py`).
3. If you need to write any additional classes needed for deserialization, please put them in the `custom_transformers.py` file. They will be imported automatically.
4. Can provide the output with any order of features.

Module 4 – Model Deployment & Monitoring (API Design & Performance Instrumentation)

Level 1

Here you will need to build a simple API that will run a regression model with given input and return the output. The API should be able to take in a JSON request with the input data and return a JSON response with the predicted class.

1. API should be runnable with `python3 app.py --port 5001` command.
2. The API should have `/predict` endpoint that takes in the HTTP POST request with JSON body, representing the input data and returns a response with the predicted value in JSON format.
3. Input json is supposed to be a dict `{"features" : [a1, a2, a3, a4, a5, a6, a7]}` where a1-a7 are numbers representing the features of the sample.
4. Response JSON is supposed to be a dictionary `{"prediction" : value}` where `value` is a model output in the form of a float number. The value should be rounded to 3 decimal places.
5. All of the requests and responses should have `Content-Type: application/json` header.
   - In case of any other content type, the API should return a JSON response with the error message in the form of dictionary `{"error" : "Content-Type not supported."}` and `415` status code.
6. In case of data was not provided in the right format the API should return a JSON response with the error message in the form of dictionary `{"error" : "<error message>"}` and `400` status code.
   - If the input data is not a dictionary, the error message should be `Input data should be a dictionary`.
   - If the input data does not have `features` key, the error message should be `Input data`

`should have "features" key.`

- If the value of features key is not a list of 7 numbers, the error message should be `Input data should have a list of 7 numbers as a value of "features" key.`

Notes.

1. Although the starting code is provided for Flask, you are allowed to use any framework you want for this question. Make sure the API has the same functionality as described in the requirements.
2. We will test the API by running `python3 app.py --port 5001` command and sending POST requests to `http://localhost:5001/predict` endpoint. Make sure your API is runnable with this command and the endpoint is correct.
3. Any process that is running on the 5001 port will be shut down before testing. If you want to test your API locally, you can run it on a different port and send requests to the endpoint with the chosen port.

Level 2

Here you need to create a simple anomaly detection system for your API. You will need to create a function that will check if the input data is an anomaly. If the input data is an anomaly, the API should return an error. Otherwise, the API should return a JSON response with the predicted value.

1. All of the requirements from level 1 should be maintained.
2. In case of anomaly, the API should return a JSON response with the error message in the form of dictionary `{“error” : “Could not process request, anomaly detected.”}` and `422` status code.
3. Data is considered an anomaly if it has at least one feature with a value outside of the range of 3 standard deviations from the mean of the respective feature in the training data.

Notes.

1. Although the starting code is provided for Flask, you are allowed to use any framework you want for this question. Make sure the API has the same functionality as described in the requirements.
2. We will test the API by running `python3 app.py --port 5001` command and sending POST requests to `http://localhost:5001/predict` endpoint. Make sure your API is runnable with this command and the endpoint is correct.
3. Any process that is running on the 5001 port will be shut down before testing. If you want to test your API locally, you can run it on a different port and send requests to the endpoint with the chosen port.