Machine Learning Engineering Core Skills Evaluation Framework

Technical Brief

CodeSignal

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 between 13-22% from 2020 to 2030[1]

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 domains into MLE, as well as growth in MLE-focused academic programs, contributing to an increasing number of new graduates entering the field. 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 for candidates.

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: 1) cleaning and manipulating data; 2) experimenting with and evaluating various models and algorithms; 3) following and incorporating research on machine learning, deep learning, and artificial intelligence; 4) solving complex problems in effective and efficient ways; and 5) understanding and incorporating key business metrics when designing machine learning product pipelines.

Although MLEs share a similar skillset as with both Data Scientists and Software Engineers, MLEs generally have stronger coding skills than Data Scientists and greater knowledge of machine learning, deep learning, and artificial intelligence than Software Engineers. This Framework, developed based on consultation with leading MLE subject matter experts, is designed specifically to assess the knowledge and skills within this domain.

---

1 Bureau of Labor Statistics, U.S. Department of Labor, Occupational Outlook Handbook, Computer and Information Technology Occupations, at https://www.bls.gov/ooh/computer-and-information-technology

Framework Specifications

The framework is designed to closely simulate the fundamental knowledge and skills a candidate would be expected to have within most Machine Learning Engineering roles across industries. The framework can be utilized to create evaluations that span across different methods of delivery, such as prescreen 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 3 modules that target coding skills (both general and specific to machine learning) and a breadth of fundamental machine learning topics. 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. The expected completion time for evaluations based on this framework is 55-75 minutes. In order to balance the depth and breadth of content and candidate experience, **the evaluation time for this framework is 70 minutes.** Possible scores range from 200 to 600.

Module 1 – Machine Learning Fundamentals

This module contains **6 scenario-based quiz questions** with an average solve time of **5-10** minutes.

Expected Knowledge

- Understanding of theories behind common machine learning algorithms, models, and concepts; typically acquired from courses that do not require research knowledge or practical experience

Can Include

- Multiple choice and fill-in-the-blank questions to measure breadth of fundamental machine learning knowledge, for example:
    - L1 vs L2 Regularization
    - Reasons for overfitting
    - Limitations of Bayes rule
    - Choosing k in a KNN algorithm
    - GBM vs Random Forest
    - Neural Network fundamentals

Should Exclude

- Complex calculations that require anything beyond a simple calculator or paper and pencil to solve
- Complex machine learning algorithms and concepts that are predominantly used in a specific sub-field (e.g., computer vision, NLP)

Module 2 – Data Manipulation

This module contains **1 basic coding question**. The average time for solving this question should be **10-15** minutes, and candidates are expected to write **15-20** lines of code.

Expected Knowledge

- Working with numbers
    - Basic operations with numbers
    - Splitting numbers into digits
- Basic string manipulation
    - Splitting a string into substrings
    - Comparing strings
    - Modifying elements of a string

    - Concatenating strings
    - Reversing a string
- Basic array manipulation
    - Iterating over an array
    - Modifying elements of an array
    - Reversing an array

Can Include

- A combination of 3 to 5 basic data structure concepts, for example:
    - Splitting a string into substrings, modifying each substring, and comparing each substring with other substrings
    - Creating two new arrays from an array given some conditions; modifying the second array and appending it to the beginning of the first array
- Requirements that are usually solvable using one or two nested loops
- Descriptions that should clearly state implementation steps

Should Exclude

- Anything that requires noticing or proving patterns
- Anything that requires optimizing algorithms
- Anything that requires knowledge of classic or niche algorithms

Module 3 – Machine Learning Algorithms Implementation

This module contains **2 coding questions** focusing on machine learning algorithms. The average time for solving these questions should be **20-25** minutes per question, and candidates are expected to write **15-25** lines of code.

Expected Knowledge

- Understanding common machine learning models and concepts
- Implementing functions and algorithms used in common machine learning models

Can Include

- Implementing a complete ML model, components of a ML model, or a ML concept based on a high-level overview of how the algorithm or concept works, for example:
    - k-Nearest Neighbors
    - k-Means Clustering
    - Decision Trees
    - Gaussian Mixture Models
    - Matrix Normalization
    - Bagging
    - Forward Propagation

Should Exclude

- Any machine learning algorithm or concept that requires pre-built libraries or packages, such as: Sklearn, Pytorch, Tensorflow, Keras
- Optimizing hyperparameters of a machine learning algorithm

Framework Example Content

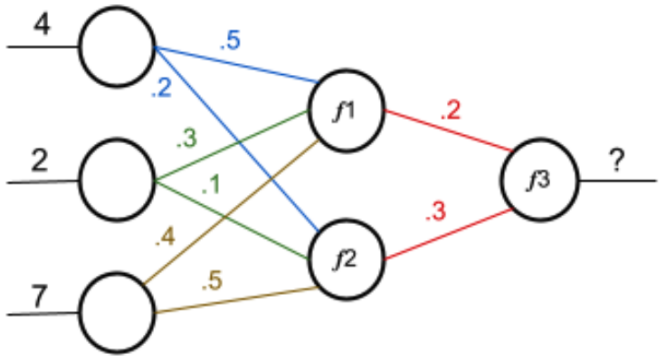
Below are example questions for each module of the framework[2]. Similar questions are consistently being developed in accordance with framework specifications and monitored on an ongoing basis to minimize the impact of potential leaks that could result in cheating or plagiarism, ensure the reliability and validity of evaluations, and provide relevant and fair candidate experiences through changing industry standards.

Module 1 – Machine Learning Fundamentals

Neural Network Fundamentals

Given the following scheme of a simple neural network, please calculate what the output value will be based on the randomly initialized weights. Please assume all bias estimates are 0, and round your answer to the nearest thousandths (three decimal places, e.g., `0.000`).

Note: Activation functions at `f1` and `f2` are **linear activation functions**, while `f3` is a **sigmoid activation function.**



Module 2 – Data Manipulation

Task

Imagine that you are working on a text classification project and part of the project requires feature engineering. One feature of interest is counting the number of words that contain triple duplicate letters in a given text string. Given a string sentence consisting of English words separated by whitespaces, count the number of words that contain triple duplicate letters – i.e., the same letter appearing at least three times within the word.

Note: letters should be counted in a case-insensitive manner (i.e., `"Pipe"` and `"pipe"` would both count as containing 2 `"p"`s). Also, consider words as any sequence of consecutive letters separated by a whitespace, which may or may not have any semantic meaning.

Example

For `sentence = "Dooddle moodle Pepper unsuccessfully"`, the output should be `solution(sentence) = 3`.

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

Explanation:

Let's take a look at all the words in this sentence:

- The word `"Dooddle"` contains a triple duplicate letter, as `"d"` appears three times;
- The word `"moodle"` does NOT contain a letter that appears three times;
- The word `"Pepper"` contains a triple duplicate letter, as `"p"` appears three times;
- The word `"unsuccessfully"` contains 2 triple duplicate letters, as both `"u"` and `"s"` appear three times.

Overall, there are `3` words in this sentence that contain triple duplicate letters, so the final answer is `3`.

Module 3 – Machine Learning Algorithms Implementation

Task

Your task is to implement parts of the k-Nearest Neighbors (kNN) algorithm from scratch (i.e., without importing any libraries or packages). The k-Nearest Neighbors classification is comprised of three major steps:

1. Calculate distance. For this task, use Euclidean Distance as the distance metric:

$$\text{EuclideanDistance} = \sqrt{\sum_{1}^{N}(x_{1i} - x_{2i})^2}$$

2. Identify k nearest neighbors
3. Assign class label by majority vote

To validate the algorithm, you will need to use it for some classification tasks. Specifically, you will be given a two-dimensional array of float values `train_data` as training data, where each subarray `train_data[i]` represents a unique case, and the last element in each subarray `train_data[i]` represents the true class label. You will also be given `test_data` as test data, with the same format as the training data (just without class labels). Assuming that k is provided, create a kNN model on the training data, use the model to classify the test data, and return class labels (float values) for the test data.

Note: It is guaranteed that all training and test data will be float values. Skeleton code for assigning and returning class labels has already been created, so please do not edit them. You should only implement code under the *`# implement this`* sections.

Skeleton Code (Python 3)

```python
# define distance metric
def euc_dist(value1, value2):
    # implement this
    pass

# identify k nearest neighbors
def k_neighbors(train_data, test_case, k):
    # implement this
    pass

# assign class labels
def get_label(train_data, test_case, k):
    neighbors = k_neighbors(train_data, test_case, k)
    labels = [row[-1] for row in neighbors]
```

```
        max_label = max(set(labels), key = labels.count)
        return max_label

    # pull it together
    def solution(train_data, test_data, k):
        final_labels = list()
        for row in test_data:
            label = get_label(train_data, row, k)
            final_labels.append(label)
        return final_labels
```

Example

For

```
train_data =   [[-2.6, 1.9, 2.0 , 1. 0, 1.0],
                [-2.8, 1.7, -1.2, 1.5, 2.0],
                [2.0, -0.9, 0.3, 2.3, 0.0],
                [-1.5, -0.1, -1.6, -1.1, 0.0],
                [-1.0, -0.6, -1.2, -0.7, 0.0],
                [-0.3, 1.2, 2.6, 0.2, 1.0],
                [-1.8, -1.3, -0.1, -1.2, 0.0],
                [0.2, 1.2, -0.6, -1.3, 1.0],
                [-5.2, 0.3, 0.2, 2.2, 2.0],
                [-0.8, -0.1, 1.5, -0.1, 0.0],
                [-2.3, 0.3, 0.8, 0.7, 2.0],
                [0.2, 3.0, 3.6,  -0.9, 1.0],
                [1.7, -0.8, -0.0, 2.0, 0.0],
                [2.8, 0.8, 1.8, -0.7, 2.0]]



test_data =    [[-0.1, 1.4, 0.4, -1.0],
                [-1.3, 0.2, -1.3, -0.8],
                [-1.1, 1.5, -2.3, -2.5],
                [0.2, 2.0, -0.1, -0.8],
                [-0.3, -1.6, -3.4, -1.4]]
```

and `k = 3`, the output should be `solution(train_data, test_data, k) = [1.0, 0.0, 0.0, 1.0, 0.0]`.

Explanation:

For each case in test_data, the kNN algorithm should:

1. Calculate its Euclidean Distance to all cases in train_data
2. Identify the k = 3 nearest cases in train_data based on Euclidean Distances, and extract their class labels
3. Assign a class label based on the most prevalent class label among the closest cases in train_data

For example, the class labels for the `k = 3` closest cases to `test_data[0]` are `train_data[5][4] = 1.0`, `train_data[7][4] = 1.0`, and `train_data[9][4] = 0.0`. So, the class label for `test_data[0] = 1.0` based on majority vote among its closest training cases. Similarly, the class labels for the `k = 3` closest cases to `test_data[1]` are `train_data[3][4] = 0.0`, `train_data[4][4] = 0.0`, and `train_data[7][4] = 1.0`. So, the class label for `test_data[1] = 0.0` based on majority vote among its closest training cases.