Case study on AI Solution

By - Jayabharathi Hari
Deakin University
: s224643593@deakin.edu.au

Case study on AI Solution - Microsoft Paper[1]

Q1.Adapting Software Engineering for Artificial Intelligence: Three Key Observations

I. **Data Discovery and Management Challenges in ML Systems:**
   A. Data Characteristics: Unlike software code (elegant, modular), data used in ML is voluminous, complex, and constantly evolving.
   B. Data Management Overhead:
      1. Finding, cleaning, processing, and storing data for training and tuning models is a significant burden.
      2. Unlike code with well-defined APIs, data often lacks explicit schema definitions, making integration and version control difficult.
      3. Rapid iteration in ML leads to frequent schema changes, further complicating data management.
      4. Updating data collection pipelines for large-scale systems can be slow and cumbersome.
      5. Versioning data effectively is challenging compared to versioning code.

II. **Customization and Reuse Challenges in ML Systems:**
   A. Difficulty in Reusing ML Models:
      1. Unlike code components (functions, libraries) that can be easily modified and reused, customizing ML models is often more complex.
      2. An ML model consists of an algorithm and learned parameters. Reusing a model effectively depends on how similar the new domain is to the training data.
      3. Significant changes in domain or input format might necessitate retraining the model, requiring additional data and expertise.
      4. Software engineers may lack the necessary ML skills to retrain or rebuild models.

III. **Modularity Challenges in ML Systems:**
   A. Limited Extensibility of ML Models:
      1. Unlike software modules that can be easily combined, extending ML models by combining pre-trained models is often not straightforward. This is because models are not easily modular and require joint training for proper interaction.
   B. Non-Obvious Interactions Between Models:
      1. In large-scale systems with multiple models, the models' outputs can influence each other's training and performance, even if their code is separate.
      2. This can lead to "component entanglement," where improvements in one model can negatively impact the overall system due to lack of adaptation in other models.

3. Incompatible updates to models can introduce new errors and break interactions with other parts of the system.

C. In essence, the passage highlights the challenges in integrating ML models into large-scale software systems due to the inherent characteristics of data in ML and the difficulties in customization, reuse, and achieving modularity with machine learning models.

Q2. Nine Stages of machine learning workflow

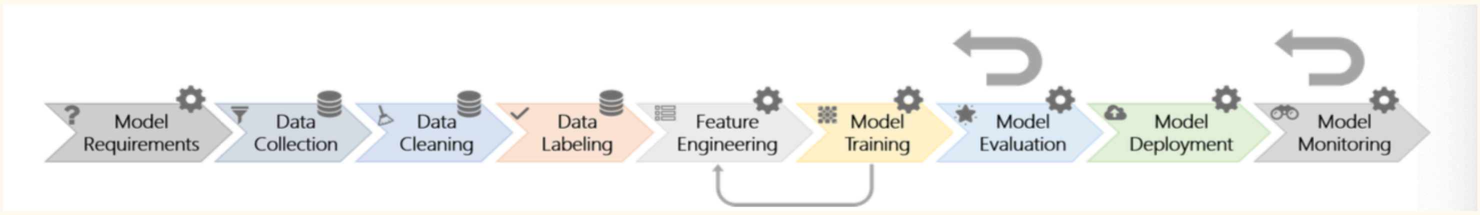

Fig. 1. The nine stages of the machine learning workflow. Some stages are data-oriented (e.g., collection, cleaning, and labeling) and others are model-oriented (e.g., model requirements, feature engineering, training, evaluation, deployment, and monitoring). There are many feedback loops in the workflow. The larger feedback arrows denote that model evaluation and monitoring may loop back to any of the previous stages. The smaller feedback arrow illustrates that model training may loop back to feature engineering (e.g., in representation learning). Ref:[2]

Machine learning projects are iterative journeys, and the success of your model hinges on following a well-defined workflow with feedback loops.

Data-Oriented Stages

1. Model Requirements:

(Model-oriented aspect) This initial stage sets the foundation for your entire project. Here, you define the problem you're trying to solve and the desired outcome of your machine learning model. This involves understanding the business need, the data you'll work with, and how you'll measure success.

2. Data Collection:

This stage focuses on gathering the raw data that will be used to train your model. There are several sources for data, depending on your project: internal databases, public datasets, web scraping (with ethical considerations), or purchased data sets.

3. Data Cleaning:

Raw data is rarely perfect. This stage focuses on identifying and correcting errors, inconsistencies, and missing values within your dataset. Common techniques include handling missing values (imputation or removal), identifying and correcting errors, and formatting the data into a usable format for machine learning algorithms.

4. Data Labeling:

Supervised learning algorithms require labeled data, where each data point has a corresponding label indicating its category or value. For example, images might be labeled as "cat" or "dog." Data labeling can be manual (time-consuming for large datasets), crowdsourced (using online platforms), or semi-supervised (using a small amount of labeled data to train an initial model for further labeling).

Model-Oriented Stages:

5. Feature Engineering:

This stage involves transforming the raw data into features that the machine learning model can effectively learn from. It's about creating meaningful representations of the data relevant to the problem you're trying to solve. Feature engineering techniques include:

- **Feature Selection:** Choosing the most relevant features that contribute to the model's performance.
- **Feature Extraction:** Creating new features by combining existing ones or using dimensionality reduction techniques.
- **Feature Scaling:** Standardizing features to a common scale for better model training.

Feedback Loop: Feature engineering often interacts with model training in a feedback loop. The initial feature set might be inadequate, requiring evaluation during training and potentially leading back to feature engineering for refinement.

6. Model Training:

This is where the model learns! You choose a suitable machine learning algorithm (e.g., decision trees, random forests, neural networks) based on your problem and data type. The algorithm learns from the prepared data, building a model that can identify patterns and relationships. Here are some key aspects:

- **Training-Validation Split:** The data is divided into training and validation sets. The model is trained on the training data and evaluated on the validation set to prevent overfitting (memorizing the data without generalizing).
- **Hyperparameter Tuning:** Adjusting the model's internal parameters to optimize its performance on the validation set.

Feedback Loop: Model evaluation heavily influences this stage. Poor performance might necessitate revisiting feature engineering, hyperparameter tuning, or even the choice of the algorithm itself.

7. Model Evaluation:

Once trained, the model's performance is evaluated on unseen data (a test set) to assess its effectiveness in real-world scenarios. Common evaluation metrics include accuracy, precision, recall, F1 score (for imbalanced datasets), and Mean Squared Error (MSE) for regression tasks.

Large Feedback Loop: Model evaluation can trigger significant rework across various stages. If the model performs poorly, you might revisit data collection (was the data representative?), data cleaning (are there errors impacting the model?), feature engineering (are the features informative?), or even the model requirements (did you define the problem correctly?).

Model-Oriented Stages (Deployment and Monitoring):

8. Model Deployment:

If the model performs well on the evaluation set, it's time to deploy it into production. This involves integrating the model into an application or system where it can be used to make real-time predictions or classifications. Deployment considerations include:

- **Scalability**: Can the model handle large volumes of data in production?
- **Performance**: Does the model meet latency and throughput requirements?
- **Explainability**: Can you understand why the model makes certain predictions?

9. Model Monitoring:

Even after deployment, the work isn't done. Monitoring the model's performance in production is crucial. This involves tracking metrics like accuracy and identifying potential issues like data drift (the data distribution changes over time, impacting the model's performance). Monitoring can trigger a feedback loop back to any of the previous stages for retraining or improvement.

By understanding these nine stages and the feedback loops that connect them, you can develop robust ML Models.

Q3. Application Of AI in Microsoft Org.

Domains using AI:

- **Traditional Areas:** Search, advertising, machine translation, customer purchase prediction, voice and image recognition.
- **Novel Areas:** Identifying customer leads, design advice for documents/presentations, unique drawing features, healthcare, improving gameplay.
- **Infrastructure Projects:** Incident reporting, bug cause identification, fraud detection, network security monitoring.

Machine Learning Approaches Used:

- **General Techniques:** Classification, clustering, dynamic programming, statistics, user behavior modeling, social network analysis, collaborative filtering.
- **Specialized Techniques:**
  - **Search:** Ranking and relevance algorithms, query understanding.
  - **Natural Language Processing (NLP):** Entity recognition, sentiment analysis, intent prediction, summarization, machine translation, ontology construction, text similarity, finding answers to questions.
  - **Finance and Sales:** Risk prediction models, forecasting.
  - **Resource Management:** Decision optimization algorithms for resource allocation, planning, pricing, bidding, and process optimization.

Q4. The best practices with ML in software engineering.

Target Business Needs with Clear Goals:

Microsoft's use of AI in areas like customer lead identification and healthcare demonstrates a focus on identifying specific business challenges where ML can deliver significant advantages.

A best practice is to clearly define the problem you're trying to solve with ML and establish measurable success criteria.

Explore a Diverse ML Toolkit:

The survey results showcase a broad spectrum of ML approaches used within Microsoft. This suggests a willingness to go beyond common algorithms and explore various techniques. A best practice is to understand the strengths and limitations of different algorithms (classification, clustering, etc.) to choose the one best suited to your specific problem and data.

Specialize When Necessary:

The paper highlights departmental specializations, like Search using ranking algorithms and NLP using sentiment analysis. This emphasizes the importance of tailoring ML approaches to specific domains. A best practice is to leverage your domain expertise and potentially develop specialized algorithms or models when needed.

Embrace Both Traditional and Novel Applications:

Microsoft utilizes AI in established areas like search and advertising, while also exploring innovative applications like design advice and improved gameplay. This approach suggests a willingness to experiment and explore new frontiers with ML. A best practice is to strike a balance between established applications (where success is more predictable) and exploring novel applications with potentially high impact.

Integrate ML Throughout Development:

The widespread adoption of AI across Microsoft teams suggests integration throughout the software development lifecycle. This implies that ML isn't an isolated step but is woven into the entire development process. A best practice is to involve data scientists and ML engineers early on to ensure a smooth integration of ML components.

Prioritize Infrastructure for ML Applications:

The use of AI for network security monitoring highlights the importance of infrastructure management for ML applications. This suggests that a robust infrastructure is crucial for successful ML deployments. A best practice is to invest in infrastructure that can handle the data storage, processing power, and model serving requirements of your ML applications.

Consider Explainability and User Trust:

While not explicitly mentioned, explainability is increasingly important in ML, especially for user-facing applications. A best practice is to consider explainability from the outset and choose techniques or models that can be interpreted to some degree, fostering user trust in the AI's decisions.

Monitor and Continuously Improve:

The success of AI projects hinges on continuous monitoring and improvement. While not explicitly stated, the use of AI for fraud detection suggests the need for ongoing monitoring to adapt to evolving threats. A best practice is to establish a feedback loop where model performance is monitored, and retraining or improvements are implemented as needed.

REFERENCES

[1]: Microsoft research website

[2]: Microsoft: Software Engineering for Machine Learning: A Case Study Paper.