VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

TRẦN NHỰT ANH - 521H0491

INTRODUCTION TO MACHINE LEARNING

UNDERGRADUATE REPORT OF COMPUTER SCIENCE

HO CHI MINH CITY, YEAR 2023

VIETNAM GENERAL CONFEDERATION OF LABOUR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

TRẦN NHỰT ANH - 521H0491

INTRODUCTION TO MACHINE LEARNING

UNDERGRADUATE REPORT OF COMPUTER SCIENCE

Advised by
Prof., LÊ ANH CƯỜNG

HO CHI MINH CITY, YEAR 2023

ACKNOWLEDGMENT

I would like to express my gratitude toward to *Prof. Lê Anh Cường* for guiding us to the completion of this report as well as your recommendations for references. *Prof. Lê Anh Cường* ‘s dedication in teaching progress has contributed well for me to complete this report.

Furthermore, I would like to acknowledge with much appreciation the learning path which Information Technology departments provided, in which supported with necessary knowledge for approaching the report topic.

Ho Chi Minh City, day 21 month 12 year 2023

Trần Nhựt Anh

DECLARATION OF AUTHORSHIP

I hereby declare that this thesis was carried out by myself under the guidance and supervision of *Prof. Lê Anh Cường*; and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by my own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

I will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

Ho Chi Minh City, day 21 month 12   year 2023

Trần Nhựt Anh

TITLE

ABSTRACT

The report’s context is mainly focused on researching the idea of optimizer in Machine Learning. General concepts of Continual Learning and Test Production in Machine Learning

Table Of Contents

1 Optimizer

1.1. Definition & Purpose

Optimizers in Machine Learning are algorithms or methods used to find the optimal set of coefficients by adjusting the parameters (or coefficients) of a model's function to improve the accuracy of it. Optimizer is a part of the Machine Learning algorithms training process. The purpose of a Machine Learning algorithm is to create a model which can predict the outcome based on given data, optimizer will maximize the model prediction accuracy.

- **Role of the Optimizer:** The primary role of an optimizer is to minimize (or in some cases, maximize) a specific function, which is usually the loss function. The loss function measures how well the model's predictions align with the actual data. By minimizing this loss, the optimizer effectively improves the model's accuracy.

- **Adjusting Model Parameters:** In the context of machine learning models like neural networks, the parameters typically include weights and biases. The optimizer adjusts these parameters using an algorithm that is based on the gradient (or derivative) of the loss function with respect to each parameter. This process is called gradient descent in its simplest form.

- **Improving Accuracy:** By iteratively updating the model's parameters, the optimizer aims to find the set of parameters that results in the lowest possible loss. A lower loss generally correlates with higher accuracy in predictions, meaning the model's outputs are more closely aligned with the actual values.

- **Types of Optimizers:** Different optimizers achieve this goal in various ways. For example:

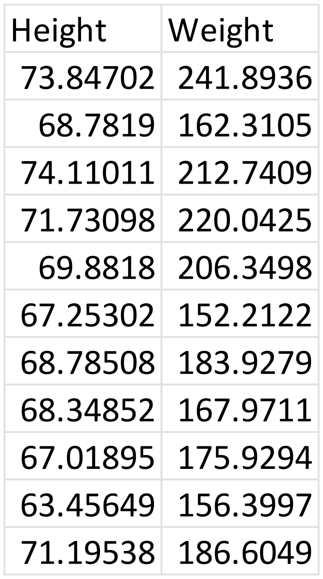
  - **Simple Gradient Descent** updates parameters in the opposite direction of the gradient.
  - **Stochastic Gradient Descent (SGD)** does this on random subsets of data, which can lead to faster convergence.
  - Advanced optimizers like Adam or RMSprop use techniques to adaptively adjust learning rates or to maintain momentum, which can lead to more efficient and effective training, especially in complex models.
- **Convergence to a Minimum:** The ultimate goal is to reach a point where the loss doesn't decrease significantly with further training. This point is often referred to as a local minimum in the loss function's surface. However, finding the global

minimum (the absolute lowest point) is often challenging, especially in high-dimensional spaces.

- **Impact on Model Performance:** The choice of optimizer and its settings (like learning rate) can significantly impact the model's performance. It's not just about accuracy; it's also about how quickly the model converges to a good solution and how stable the training process is.

1.2. Gradient Descent

For example, Given a dataset like this. From this train a model which can predict Height based on Weight



Height	Weight
73.84702	241.8936
68.7819	162.3105
74.11011	212.7409
71.73098	220.0425
69.8818	206.3498
67.25302	152.2122
68.78508	183.9279
68.34852	167.9711
67.01895	175.9294
63.45649	156.3997
71.19538	186.6049

Figure 1: Sample Data

The dataset will be represented as $(x_1, y_1) \dots \dots (x_d, y_d)$

From this dataset we can visualize it on a xy graph with x is the weight and y are the height as well as the initial model for predicting the Height from Weight. $y = bias + w * x$ The initial function parameter for bias and w is 0 and 1. Notice that all the dataset has been preprocessed to range 0 to 10.

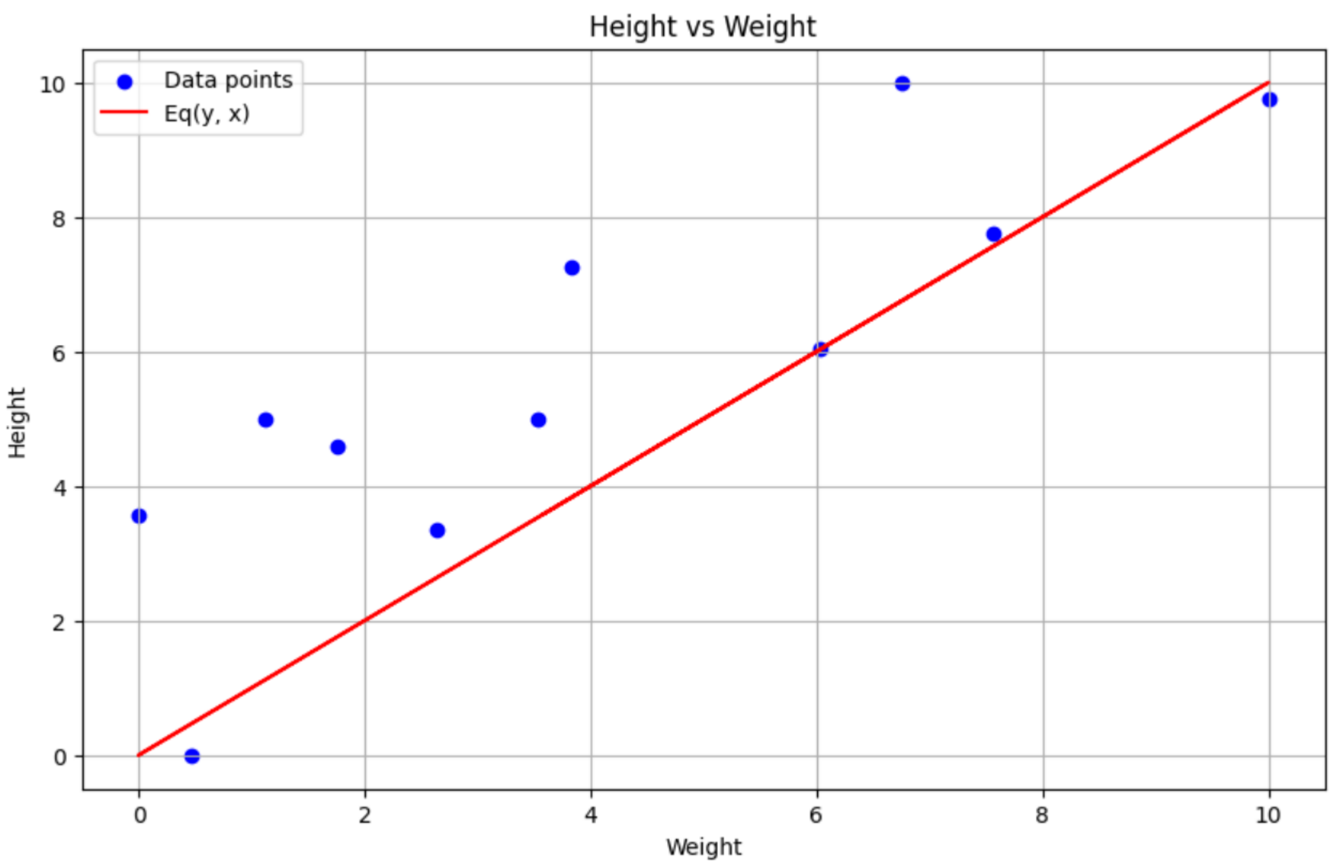


Figure 2 : Initial Model's Function

From this initial model's function, we process the optimizer specifically gradient descent here. With the following steps:

1. **Calculating the Loss:** Typically, the mean squared error (MSE) is used as the loss function in regression problems. It measures the average squared difference between the predicted and actual values.

In this example, with the $y = x$ function gives out $\hat{y}$ which the predict values from x. The MSE is calculated as: $MSE = \frac{1}{d} \sum_{i=1}^{d} (y_i - \hat{y}_i)^2$. This MSE transforms to a Loss function for simplicity purposes of computing gradients. Therefore, The Loss function for computing gradients is:

$$Loss = L = \frac{1}{2d} \sum_{i=1}^{d} ((w_0 - w_1 x_i) - y_i)^2$$

2. **Computing the Gradients (or derivates):** The partial derivatives of the loss function with respect to bias and weight are calculated. These gradients indicate the direction to move in the parameter space to reduce the loss.

From the Loss Function above, compute the gradients by derivate, the Loss Function with respect to w_0 and w_1 in this example, if the Loss Function have more w parameters, then the number of derivatizations will be corresponding to the number of parameters,

$$L' = \frac{1}{2d}\sum_{i=1}^{d}((w_0 - w_1x_i) - y_i)^2 \frac{dL}{dw_0}$$

$$\frac{\partial L}{\partial w_0} = \frac{1}{d}\sum_{i=1}^{d}(w_0 - w_1x_i - y_i) \qquad GOAL\ minize\ to\ \frac{\partial L}{\partial w_0} = 0\,, \frac{\partial L}{\partial w_1} = 0$$

$$y = \frac{1}{2d}\sum_{i=1}^{d}((w_0 - w_1x_i) - y_i)^2 \frac{dy}{dw_1}$$

$$\frac{\partial L}{\partial w_1} = \frac{1}{d}\sum_{i=1}^{d} x_i\,(w_0 - w_1x_i - y_i)$$

3. **Updating the Parameters:** The bias and weight are updated in the opposite direction of the gradients, scaled by a learning rate (μ).

$$w_i' = w_i - \mu\frac{\partial L}{\partial w_i}$$

In this example,

$$w_0' = w_0 - \mu\frac{\partial L}{\partial w_0}$$
$$w_1' = w_1 - \mu\frac{\partial L}{\partial w_1}$$

From this update the W parameter

4. **Iterating:** These steps are repeated for several iterations or until the loss converges to a minimum value. As the loss converges to minimum value the change in w_i' the difference between w_i' and w_i is not significant as step is shorter.

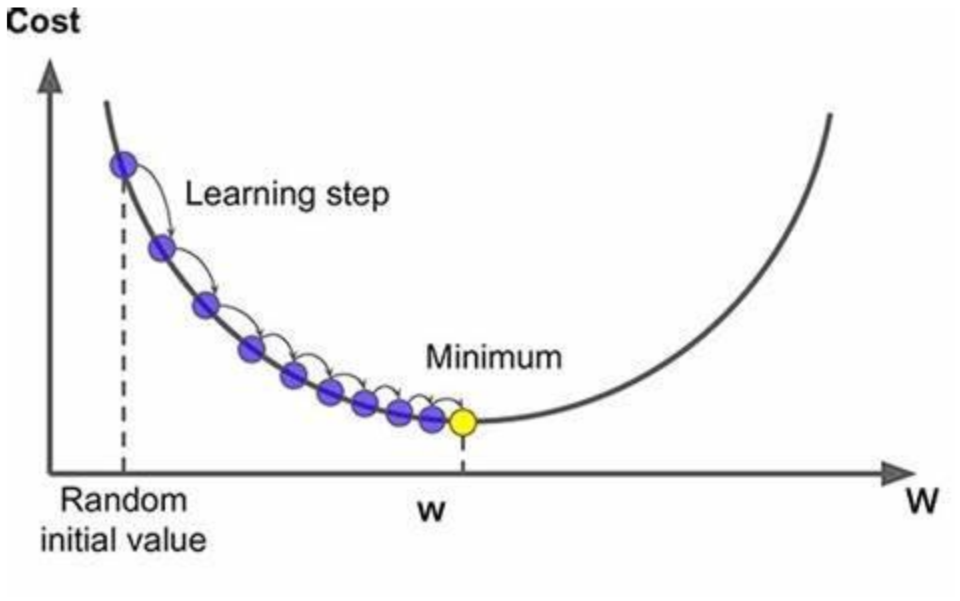


Figure 3: Cost Function Based On W

From this simple example with one feature each datapoint problem arises as the datapoint begins to increase in term of numbers of features and dataset size increases. As each iteration (or epoch) the calculation is more complex which makes the computational to be expensive and slow.

In the example each datapoint only has 2 variables (or features) which make the loss function convex. This means that there is only one minimum: the global minimum. Therefore, there are no local minima or saddle points to worry about, and gradient descent is guaranteed to converge to the global minimum, given enough time and an appropriate learning rate.

In a local minimum, the gradient of the loss function is zero. If you are using batch gradient descent (which uses the entire dataset to compute the gradient), the model parameters will no longer be updated once a local minimum is reached, potentially leading to suboptimal solutions.

However, as the complexity of the model increases (with more features or non-linearities), the loss surface can become non-convex, and local minima and saddle points can appear. This is particularly true in models such as neural networks, where the loss function is a high-dimensional surface with a complex landscape.

- **Local Minima:** These are points in the parameter space where the loss is lower than in the immediate vicinity but not necessarily the lowest possible loss value (which would be the global minimum). In a non-convex function, there can be many local minima.

- **Saddle Points:** These are points where the gradient is zero (like in minima or maxima), but they are not local minima in all dimensions. Along some dimensions, they can be a maximum. In high-dimensional spaces, saddle points can be more common than local minima.

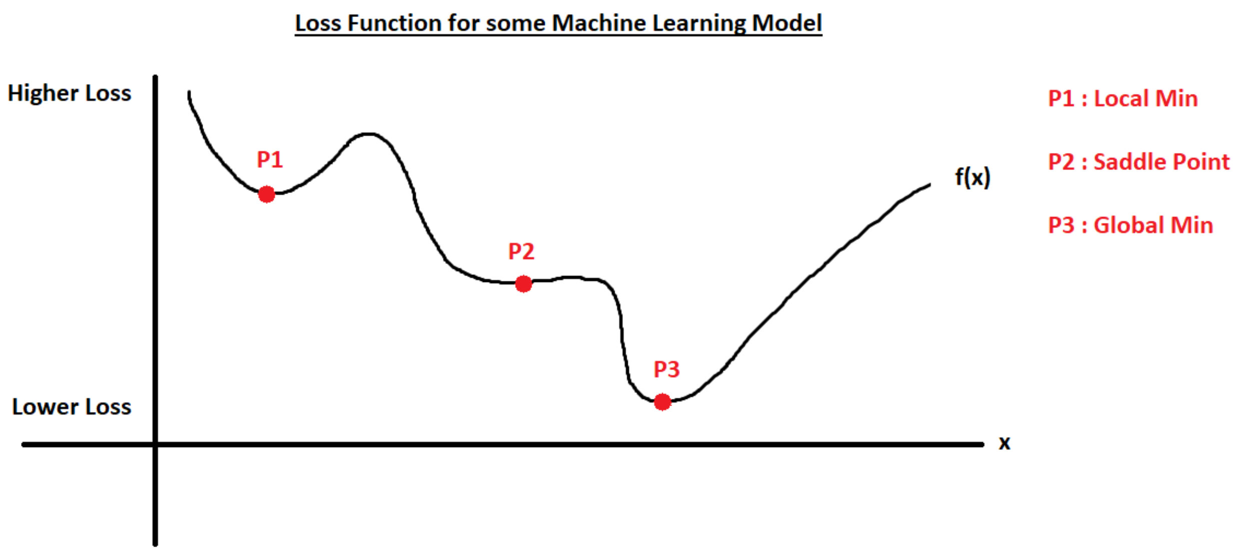


Figure 4 : Loss Function in Complex Models

1.3.Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) follows the core principles of Gradient Descent (GD) in optimizing model parameters to minimize a loss function. However, SGD differs significantly in its approach to computing and applying updates to the model parameters.

Definition and Process:

- SGD Algorithm: In SGD, the model parameters are updated for each training example xi and label yi. Rather than summing the error over all examples (as in GD), SGD updates the parameters based on the error of a single data point at each iteration.

- Loss Function: For a given model with parameters w0 (bias) and w1 (weight), and a dataset of features and labels, the loss for a single data point can be represented as:

$$Loss = L = \frac{1}{2d}\left((w_0 - w_1 x_i) - y_i\right)^2$$

With this loss function the remaining process will be familiar to the Gradient Descent process mentioned above.

However, the simplicity of this method is susceptible to datapoint which feature is too excluded compared to remain leading to not produce the most optimal model. Because at a specific epoch only a datapoint is process causing the updated parameters to be less accurate than the previous. This makes the convergence path less stable.

- In SGD, the gradient is computed using one data point (or a small batch) at a time. This means the gradient is an approximation based on a subset of the data.
- If the parameters are at a local minimum with respect to the entire dataset, they might not be at a minimum for an individual data point or a small batch. Therefore, the computed gradient for that point or batch might not be zero.
- This non-zero gradient leads to an update in the parameters, potentially moving them out of the local minimum when considering the entire dataset.

1.4. Momentum

Momentum is an optimization technique that can be applied to Gradient Descent (including Stochastic Gradient Descent) to speed up convergence, particularly in scenarios involving high curvature, small but consistent gradients, or noisy gradients. Let's discuss how Momentum works in the context of a cost function with two variables.

Basic Concept:

- Idea: Momentum helps in accelerating the gradient descent algorithm by considering the past gradients. It is inspired by the physical concept of momentum in mechanics, where it keeps an object moving in a particular direction.
-

Applying Momentum to a Two-Variable Cost Function example:

$$w_0' = w_0 - \mu \frac{\partial L}{\partial w_0}$$
$$w_1' = w_1 - \mu \frac{\partial L}{\partial w_1}$$

When Momentum is applied, 2 new variables, ϑ_{w_0} and ϑ_{w_1} which is define as.

$$\vartheta_{w_0} = \beta\vartheta_{w_0} + (1-\beta)\frac{\partial L}{\partial w_0}$$
$$w_0' = w_0 - \mu\vartheta_{w_0}$$

$$\vartheta_{w_1} = \beta\vartheta_{w_1} + (1-\beta)\frac{\partial L}{\partial w_1}$$
$$w_1' = w_1 - \mu\vartheta_{w_1}$$

- Momentum Coefficient (β) This is a new hyperparameter (usually set between 0.9 and 0.99). It determines how much of the previous velocity is retained. A higher β means more of the past gradients are considered, leading to a smoother and more stable update path.

- Velocity Updates: The 'velocity' for each parameter is a weighted average of its previous velocity and the current gradient. This means each update considers not just the current gradient but also the direction and magnitude of past updates.

- Parameter Updates: Parameters are updated using this velocity, which can lead to faster convergence, especially in scenarios where the surface of the cost function has ravines or where gradients are consistently pointing in a particular direction.

➢ Benefits of Momentum:
- Smoothing Effect: Momentum has a smoothing effect on the optimization path, which helps to navigate the cost function more effectively.
- Faster Convergence: It can accelerate convergence, particularly in the presence of small gradients or gradients that don't change direction frequently.
- Navigating Ravines and Plateaus: Momentum is particularly useful in scenarios where the cost function has ravines (areas where the surface curves much more steeply in one dimension than in another) or plateaus (areas of low gradient).

➢ Visualization:

In a two-variable cost function, imagine a ball rolling down the surface of the cost function. Without momentum, the ball strictly follows the steepest descent path. With momentum, the ball's velocity helps it to continue moving in the same direction as before, which can help it roll through narrow ravines and over plateaus more effectively.

In summary, momentum adds a form of memory to the optimization process, considering past gradients in the updates, leading to faster and more stable convergence in many scenarios.

1. Continual Learning and Test Production in Machine Learning

1.1 Continual Learning

Continual Learning (Preetipadma, 2020), also known as Lifelong Learning, is a paradigm in machine learning where the model is designed to learn continuously, accumulating the knowledge gained over time and adapting to new tasks or changes in the data distribution. This approach contrasts with traditional machine learning models, which are typically trained once on a fixed dataset and not expected to adapt to new information.

- **Key Concepts**

- Incremental Learning: Continual learning involves incrementally updating the model's knowledge without needing to retrain from scratch on the combined old and new data.

- Transfer Learning: It often leverages transfer learning principles, where knowledge gained while learning one task is applied to improve learning of subsequent tasks.

- Avoiding Catastrophic Forgetting: A major challenge in continual learning is catastrophic forgetting, where the model forgets previously learned information upon learning new data. Effective strategies are essential to mitigate this.

- Task Agnostic Learning: Ideally, continual learning systems can learn from a stream of data without explicit task boundaries, making them more robust and adaptable.

Applications

- **Robotics:** Adapting to new environments or tasks without human intervention.
- **Personalized Recommendations:** Updating recommendations based on evolving user preferences.
- **Autonomous Vehicles:** Adapting to new driving conditions or regulations over time.

Strategies for Continual Learning

- Regularization Techniques: Methods like Elastic Weight Consolidation (EWC) apply constraints on the update of parameters important for previous tasks.

- Architectural Approaches: Designing models that dynamically expand with new tasks, allocating separate parameters for different tasks to avoid interference.

- Replay Mechanisms: Storing a subset of previous data or generating pseudo-data to replay old tasks while learning new tasks.

- Hybrid Approaches: Combining various strategies to balance the retention of old knowledge with the acquisition of new information.

Challenges and Future Directions

- Memory Efficiency: Developing methods that require minimal memory overhead for storing past information.
- Task-Free Learning: Creating models that can continually learn without clear task delineations.

- Scalability: Ensuring continual learning approaches scale effectively to real-world scenarios with complex and high-dimensional data.

Continual learning is a promising direction in machine learning (Continuous Machine Learning: Why is it important ?, 2022), aiming to create models that more closely resemble human learning by adapting and accumulating knowledge over time. As research progresses, it has the potential to significantly impact various domains, leading to more flexible and intelligent systems.

1.2 Test Production

Testing in production, also known as "production testing," is an important phase in the machine learning (ML) lifecycle. It refers to the process of evaluating the performance of ML models in a live environment where real-world, dynamic data is encountered. This phase is crucial for understanding how a model performs in practical scenarios, beyond controlled test environments.

Key Concepts of Test Production in Machine Learning:

- **Real-World Data:** Testing in production exposes the model to real-world data that might not have been available or considered during the training phase.

- **Performance Monitoring:** Continuous monitoring is essential to track the model's performance in production. Metrics like accuracy, precision, recall, and others specific to the model's task are typically monitored.

- **Feedback Loop:** Production testing often involves a feedback loop where predictions are compared against actual outcomes (when they become available) to assess and continually improve the model's performance.

- **Model Drift and Retraining:** Over time, models in production can suffer from "model drift" as data patterns change. Regular monitoring helps in identifying when a model needs to be retrained or fine-tuned.

- **A/B Testing:** Deploying different versions of models simultaneously to subsets of users (A/B testing) can provide insights into which model performs better in a live setting.

- **Canary Releases:** Gradually rolling out a new model to a small percentage of the user base before a full deployment helps in mitigating risks.

Challenges in Test Production:

- **Data Skew and Shift:** Models might encounter data that significantly differs from the training data, leading to performance degradation.
- **Scalability and Resource Management:** Ensuring that the model scales efficiently with the increase in data and user load, while managing computational resources effectively.
- **Error Analysis and Debugging:** Identifying the root causes of performance issues in a production environment can be more complex compared to controlled test environments.
- **Regulatory and Ethical Considerations**: In certain domains (like healthcare or finance), models in production must comply with regulatory standards and ethical guidelines.

Best Practices:

- **Robust Logging and Monitoring:** Implement comprehensive logging of model inputs, predictions, and performance metrics for ongoing evaluation and troubleshooting.
- **Failover and Redundancy Strategies:** Have strategies in place for model failure, including fallback mechanisms.
- **Continuous Learning and Updating:** Implement mechanisms for the model to learn from new data and update periodically to maintain its relevance.
- **User Feedback Integration:** Incorporate user feedback into the model evaluation process.
- **Ethical and Fairness Checks:** Regularly test for biases and fairness issues in the model's predictions.

Test production is a critical stage in the ML lifecycle, ensuring that models are not only theoretically sound but also practically effective in real-world scenarios. It requires careful planning, monitoring, and a proactive approach to maintenance and improvement.

2 Bibliography

Continuous Machine Learning: Why is it important ? (2022, 5 5). Retrieved from Task Us: https://www.taskus.com/insights/continuous-machine-learning/

Preetipadma. (2020, 10 20). *Continual Learning: An Overview into the Next stage of AI*. Retrieved from Analytics Insight: https://www.analyticsinsight.net/continual-learning-an-overview-into-the-next-stage-of-ai/