VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**


# FINAL REPORT
# MACHINE LEARNING

*Instructor:* **GV LE ANH CUONG**

*Student's name:* **NGUYỄN THANH BẢO -520H0514**

*Class:* **20H50205**

*Course:* **24**

**HO CHI MINH CITY, 2023**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**


# FINAL REPORT
# MACHINE LEARNING

*Instructor:* **GV LE ANH CUONG**

*Student's name:* **NGUYỄN THANH BẢO -520H0514**

*Class:* **20H50205**

*Course:* **24**

**HO CHI MINH CITY, 2023**

# ACKNOWLEDGEMENTS

After half a semester of studying at Ton Duc Thang University, we want to give our sincere thanks to our teachers and Faculty of Information Technology for bringing a desired condition for students able to complete their course by giving many topics that can be applied in real-life.

We want to express our gratitude for our theory lecturer – Le Anh Cuong. He helped us a lot on this course, giving us information about Machine Learning so that we can improve our knowledge. Also, we want to give thanks to each other member in this group. All of us are busy because everyone has other classes, but we still reply to each other when we need to. Once again, we are truly grateful for everyone that helped us to do this project. With challenging work and effort, we have successfully completed this report. Maybe this report has some mistakes because this is our first report on this course, so we really want to take the comments of the teacher and lecturer for our improvement in future projects.

With sincere thanks.

# THE REPORT WAS COMPLETED AT TON DUC THANG UNIVERSITY

We assure this is our own project with the instruction of Le Anh Cuong. All the researches, the results in this report are trustworthy and have never been announced in any appearance before. The data in tables for analysis, comments, evaluations were collected by the student in many different sources, which have clearly written in references.

Besides, we used some comments, evaluations, analysis and data of other writer, organizations in the project – which is also in the citations and source notes.

**If there is any fraud in my project, we will take full responsibility for our report content.** Ton Duc Thang University is not related to the copyright infringement that we made during the implementation process (if available).

*Ho Chi Minh City, Dect* 23rd*, 2023*

*Authors*

*(Full name and signature)*

*Nguyễn Thanh Bảo*

# EVALUATION OF INSTRUCTING LECTURER

**Confirmation of the instructor**

_____
_____
_____
_____
_____
_____
_____

*Ho Chi Minh City, 2023*

*(Sign and provide full name)*

**The assessment of the teacher marked**

_____
_____
_____
_____
_____
_____
_____

*Ho Chi Minh City, 2023*

*(Sign and provide full name)*

# TABLE OF CONTENTS

# CHAPTER 1 – OPTIMIZER IN MACHINE LEARNING MODELS

## 1.1 INTRODUCTION TO MODEL OPTIMIZATION ALGORITHM

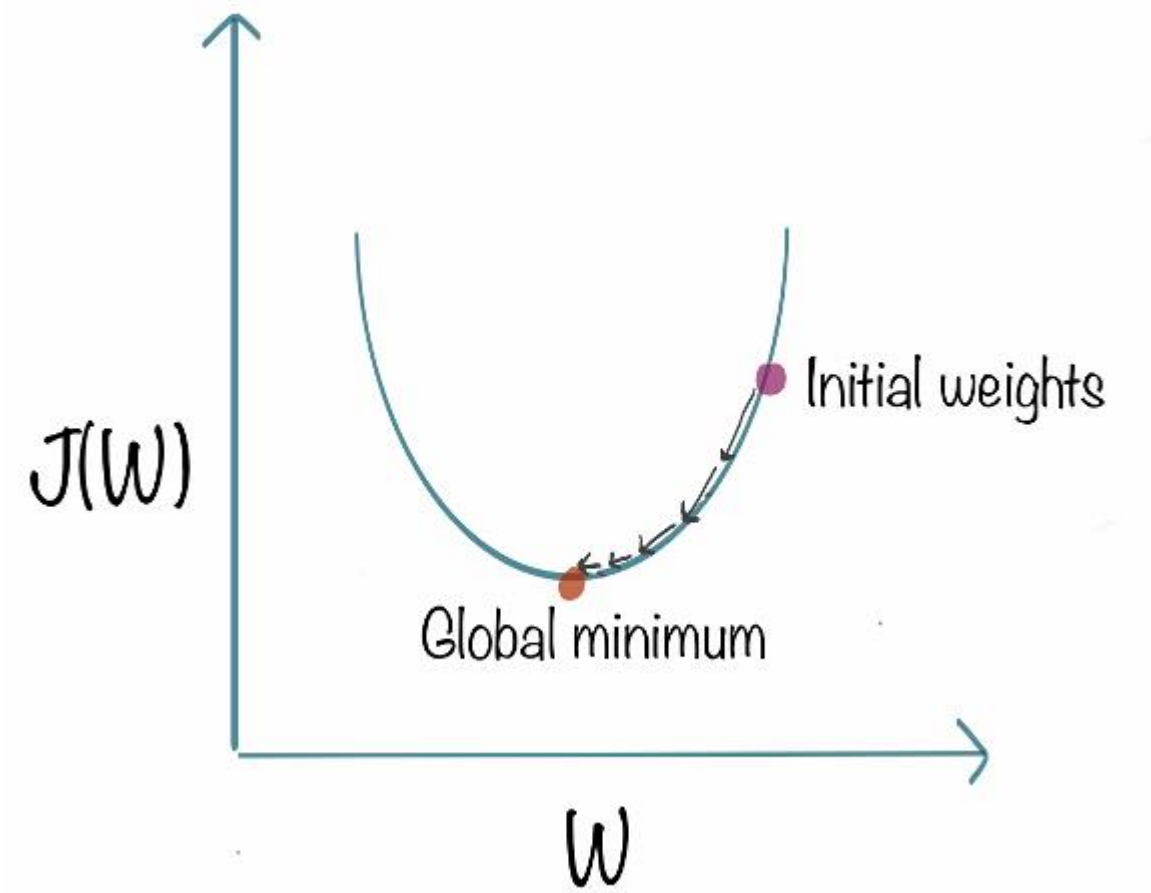
Optimizers are algorithms that adjust the model's parameters during training to minimize a loss function. They enable neural networks to learn from data by iteratively updating weights and biases. Common optimizers include Stochastic Gradient Descent (SGD), Adam, and RMSprop. Each optimizer has specific update rules, learning rates, and momentum to find optimal model parameters for improved performance.

Optimizer algorithms are optimization method that helps improve a model's performance. These optimization algorithms or optimizers widely affect the accuracy and speed training of the deep learning model

## 1.2 GRADIENT DESCENT OPTIMIZER

This optimization algorithm uses calculus to modify the values consistently and to achieve the local minimum.
Gradient descent works as follows:

- It starts with some coefficients, sees their cost, and searches for cost value lesser than what it is now.
- It moves towards the lower weight and updates the value of the coefficients.
- The process repeats until the local minimum is reached. A local minimum is a point beyond which it cannot proceed




Gradient descent works best for most purposes. However, it has some downsides too. It is expensive to calculate the gradients if the size of the data is huge. Gradient descent works well for convex functions, but it doesn't know how far to travel along the gradient for nonconvex functions.

### 1.3 STOCHASTIC GRADIENT DESCENT OPTIMIZER

Using gradient descent on massive data might not be the best option. To tackle the problem, we have stochastic gradient descent. The term stochastic means randomness on which the algorithm is based upon. In stochastic gradient descent, instead of taking the whole dataset for each iteration, we randomly select the batches of data. That means we only take a few samples from the dataset.

$$w := w - \eta \nabla Q_i(w).$$

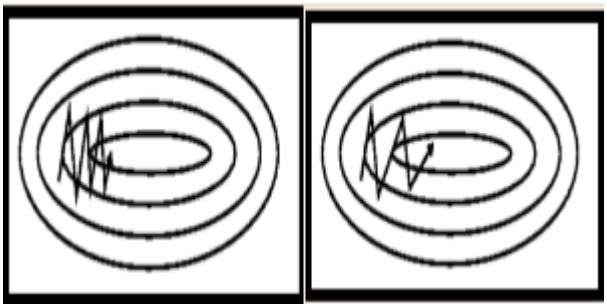
The procedure is first to select the initial parameters w and learning rate n. Then randomly shuffle the data at each iteration to reach an approximate minimum.

Since we are not using the whole dataset but the batches of it for each iteration, the path taken by the algorithm is full of noise as compared to the gradient descent algorithm. Thus, SGD uses a higher number of iterations to reach the local minima. Due to an increase in the number of iterations, the overall computation time increases. But even after increasing the number of iterations, the computation cost is still less than that of the gradient descent optimizer. So the conclusion is if the data is enormous and computational time is an essential factor, stochastic gradient descent should be preferred over batch gradient descent algorithm.

## 1.4 STOCHASTIC GRADIENT DESCENT WITH MOMENTUM OPTIMIZER

We use stochastic gradient descent with a momentum algorithm to overcome huge number of iterations to reach optimal minimum.

It helps in faster convergence of the loss function. Stochastic gradient descent oscillates between either direction of the gradient and updates the weights accordingly. However, adding a fraction of the previous update to the current update will make the process a bit faster, and learning rate should be decreased with a high momentum term.



In the above image, the left part shows the convergence graph of the stochastic gradient descent algorithm. At the same time, the right side shows SGD with momentum. From the image, you can compare the path chosen by both algorithms and realize that using momentum helps reach convergence in less time.

## 1.5 MINI BATCH GRADIENT DESCENT

Instead of taking all the training data, only a subset of the dataset is used for calculating the loss function. Since we are using a batch of data instead of taking the whole dataset, fewer iterations are needed. That is why the mini-batch gradient descent algorithm is faster than both stochastic gradient descent and batch gradient descent algorithms. This algorithm is more efficient and robust than the earlier variants of gradient descent. As the algorithm uses batching, all the training data need not be loaded in the memory, thus making the process more efficient to implement. Moreover,

the cost function in mini-batch gradient descent is noisier than the batch gradient descent algorithm but smoother than that of the stochastic gradient descent algorithm. Because of this, mini-batch gradient descent is ideal and provides a good balance between speed and accuracy.
It needs a hyper parameter that is “mini-batch-size”, which needs to be tuned to achieve the required accuracy. Although, the batch size of 32 is considered to be appropriate for almost every case. Also, in some cases, it results in poor final accuracy.

**1.6 ADAGRAD (ADAPTIVE GRADIENT DESCENT)**
Adagrad uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training. The more the parameters get changed, the more minor the learning rate changes. This modification is highly beneficial because real-world datasets contain sparse as well as dense features. The Adagrad algorithm uses the below formula to update the weights. Here the alpha(t) denotes the different learning rates at each iteration, n is a constant, and E is a small positive to avoid division by 0.

$$\mathrm{w_t} = \mathrm{w_{t-1}} - \eta'_{\mathrm{t}} \frac{\partial \mathrm{L}}{\partial \mathrm{w(t-1)}}$$

$$\eta'_{\mathrm{t}} = \frac{\eta}{\mathrm{sqrt}(\alpha_{\mathrm{t}} + \epsilon)}$$

The benefit of using Adagrad is that it abolishes the need to modify the learning rate manually. It is more reliable than gradient descent algorithms and their variants, and it reaches convergence at a higher speed.

One downside of the AdaGrad optimizer is that it decreases the learning rate aggressively and monotonically. There might be a point when the learning rate becomes extremely small. This is because the squared gradients in the denominator keep accumulating, and thus the denominator part keeps on increasing. Due to small learning rates, the model eventually becomes unable to acquire more knowledge, and hence the accuracy of the model is compromised.

**1.7 RMS PROP (ROOT MEAN SQUARE)**
This algorithm uses the gradient sign, adapting the step size individually for each weight. In this algorithm, the two gradients are first compared for signs. If they have the same sign, we are going in the right direction, increasing step size. If opposite

signs, decrease step size. Then we limit the step size and can now go for the weight update.
The problem with RPPROP is that it doesn't work well with large datasets and when we want to perform mini-batch updates. RMS prop is an advancement in AdaGrad optimizer as it reduces the monotonically decreasing learning rate.

**Formula:**
The algorithm mainly focuses on accelerating the optimization process by decreasing the number of function evaluations to reach the local minimum. The algorithm keeps the moving average of squared gradients for every weight and divides the gradient by the square root of the mean square.

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

where gamma is the forgetting factor. Weights are updated by the below formula

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

In simpler terms, if there exists a parameter due to which the cost function oscillates a lot, we want to penalize the update of this parameter. This algorithm has several benefits as compared to earlier versions of gradient descent algorithms. The algorithm converges quickly and requires lesser tuning than gradient descent algorithms and their variants.

The problem with RMS Prop is that the learning rate has to be defined manually, and the suggested value doesn't work for every application.

**1.8 ADAM (ADAPTIVE MOMENT ESTIMATION**
It is an extension of the stochastic gradient descent algorithm and is designed to update the weights of a neural network during training.

Unlike stochastic gradient descent, which maintains a single learning rate throughout training, Adam optimizer dynamically computes individual learning rates based on the past gradients and their second moments.

Similar to RMSProp, Adam optimizer considers the second moment of the gradients, but unlike RMSProp, it calculates the uncentered variance of the gradients (without subtracting the mean).
By incorporating both the first moment (mean) and second moment (uncentered variance) of the gradients, Adam optimizer achieves an adaptive learning rate that can

efficiently navigate the optimization landscape during training. This adaptivity helps in faster convergence and improved performance
In summary, Adam optimizer is an optimization algorithm that extends stochastic gradient descent by dynamically adjusting learning rates based on individual weights. It combines the features of AdaGrad and RMSProp to provide efficient and adaptive updates to the network weights during deep learning training.
**Formula:**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[ \frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[ \frac{\delta L}{\delta w_t} \right]^2$$

The above formula represents the working of adam optimizer. Here B1 and B2 represent the decay rate of the average of the gradients.

Adam has some downsides. It tends to focus on faster computation time, whereas algorithms like stochastic gradient descent focus on data points. That's why algorithms like stochastic gradient descent generalize the data in a better manner at the cost of low computation speed. So, the optimization algorithms can be picked accordingly depending on the requirements and the type of data.

# CHAPTER 2 – CONTINOUS LEARNING AND TEST PRODUCTION

## 2.1 CONTINOUS LEARNING

Continuous learning, is a process in which a model learns from new data streams without being re-trained.

Contrary to traditional approaches, where models are trained on a static dataset, deployed, and periodically re-trained, continuous learning models iteratively update their parameters to reflect new distributions in the data.

In the latter process, the model improves itself by learning from the latest iteration and updating its knowledge as new data becomes available. The continuous learning model life-cycle enables models to remain relevant over time due to their inherently dynamic quality.

### 2.1.1 Types of continuous learning:

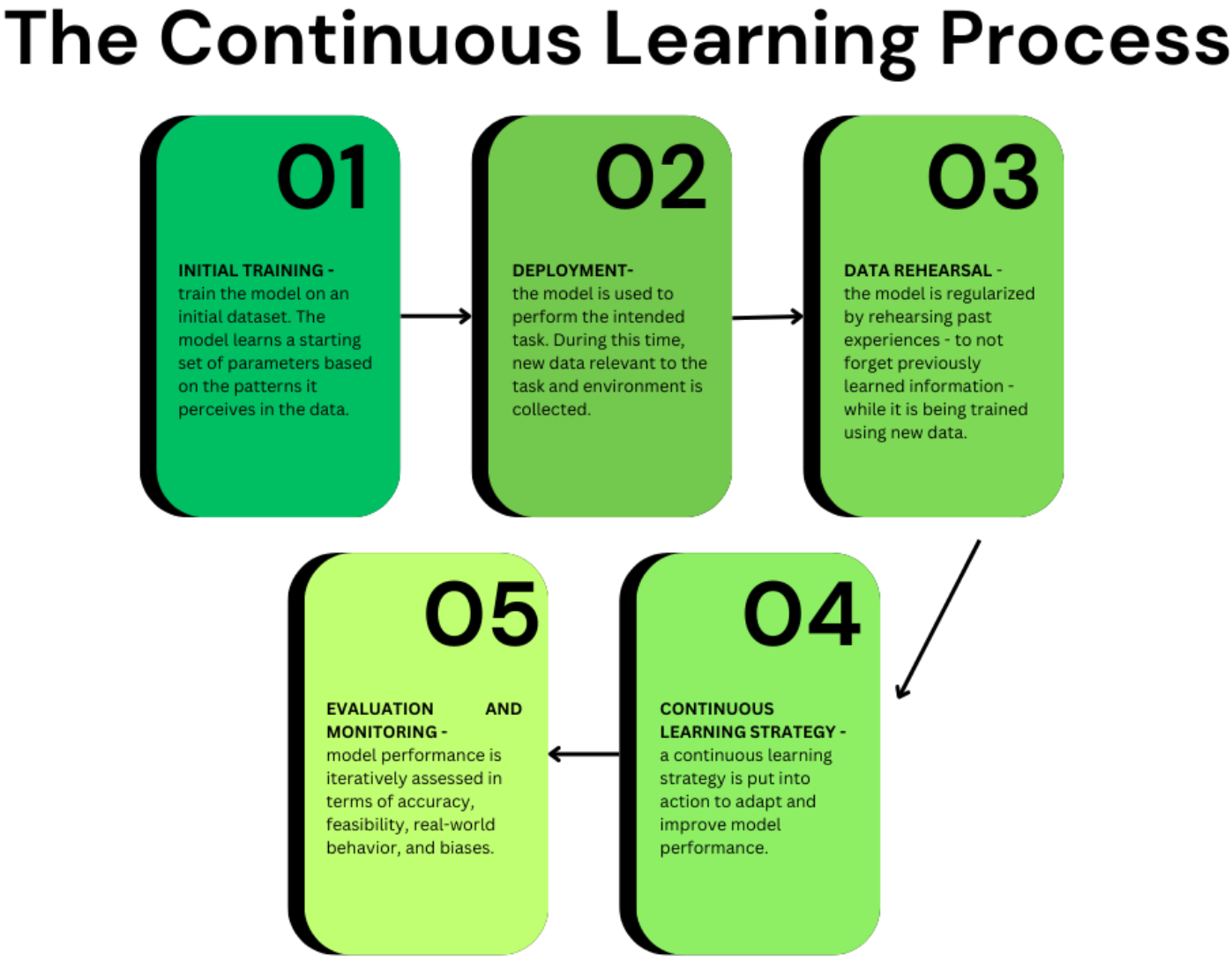
There are multiple continuous machine learning approaches to modeling. Popular strategies include incremental learning, transfer learning, and lifelong learning. Other examples are experience replay methods and regularization techniques.

Like with all things data, the choice of approach is not black and white. Instead, it depends on the data, model architecture, desired performance, task complexity, and computational resources available. Oftentimes, a combination of approaches is implemented to enhance learning capabilities.

### 2.1.2 The Continuous Learning Process

Continuous learning is an evolution of traditional machine learning modeling. Therefore, it involves many of the same modeling principles: pre-processing, model selection, hyper parameter tuning, training, deployment, and monitoring.

Two additional steps are required in the continuous learning process: data rehearsal and the implementation of a continuous learning strategy. These steps serve to ensure that the model is learning from streams of new data efficiently based on the application and context of the data task.




### 2.1.3 Advantages of Continuous Learning

Continuous learning can be useful for all types of data projects: descriptive, diagnostic, predictive, and prescriptive. It is particularly important in cases involving fast-changing data. Benefits compared to a traditional machine learning approach include:

- Generalization. Continuous learning empowers the model to be more robust and accurate in the face of new data.

- Retention of information. By employing a continuous learning strategy, the model considers previous knowledge gained in past iterations, enabling it to accumulate information over time.
- Adaptability. A model employing continuous learning adapts to new knowledge – such as concept drift and new trends – thereby having greater predictive capabilities in the long run.

### 2.1.4 Limitations of Continuous Learning

**Cost**: Continuous learning approaches, while effective, also tend to be more computationally complex than traditional ones as the model needs to consistently adapt to new data. Said complexity often translates into higher economic costs because it necessitates more data, human, and computing resources.

**Model management**: Every time a model's parameters update based on new data, a new model is formed. Therefore, a continuous learning approach may generate a large number of models, complicating the identification of best-performing ones.

**Data drift**: For a continuous learning approach to be worthwhile, we must process a large volume of new data. However, such a model risks the chance of losing predictive capabilities if the feature distribution changes abruptly. Learn more about data drift in a separate article.

While cost is not a limitation easy to circumvent, challenges related to modeling can be alleviated by having a proper methodology in place and through human intervention. Practices such as model versioning, monitoring, and evaluation are key to tracking model performance. In addition, human intervention is important to enforce the practices mentioned above and to make contingent choices about the data, such as the frequency and size of updates.

### 2.1.5 Applications of Continuous Learning

Current applications of continuous learning include:

- Computer vision. The dynamic nature of image-driven data makes continuous learning approaches popular for training algorithms to identify and classify visual information. These are frequently applied in facial recognition and imaging technology.
- Cybersecurity. Continuous learning approaches are implemented to ensure constant monitoring in IT security infrastructures. They are key in detecting phishing, network intrusion, and spam, among other security-related operations.

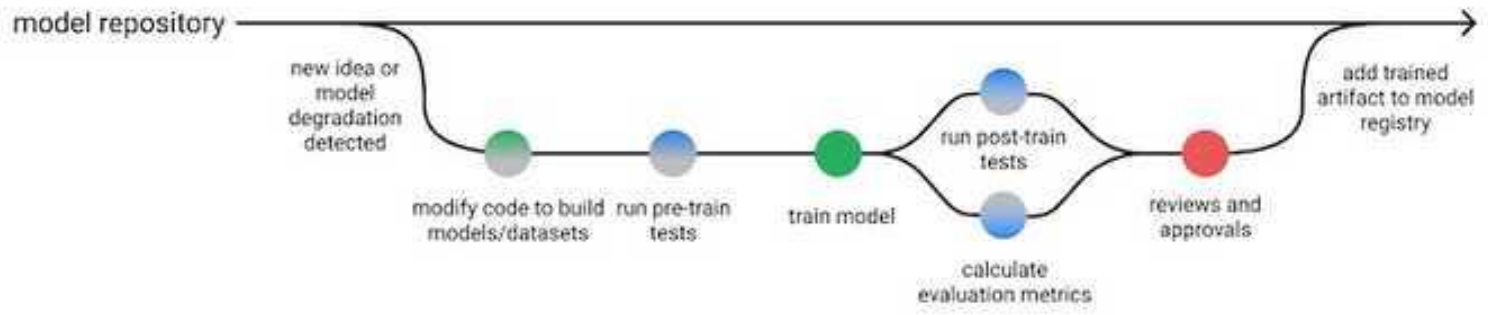
- Healthcare. Due to the evolving essence of diseases, continuous learning approaches are used in various healthcare fields to enhance diagnostic workflows around disease diagnosis. Specializations such as oncology and radiology have been early movers in exploring continuous learning AI to this end.
- Robotics. Continuous learning has been utilized to enhance the adaptability and performance of robots in different environments, enabling them to optimize their actions in changing conditions based on past and new experiences.

## 2.2 TEST PRODUCTION

Machine learning tests can be distinguished between traditional software tests and machine learning (ML) tests; software tests check the written logic while ML tests check the learned logic.

ML tests can be further split into testing and evaluation. We're familiar with ML evaluation where we train a model and evaluate its performance on an unseen validation set; this is done via metrics (e.g., accuracy, Area under Curve of Receiver Operating Characteristic (AUC ROC)) and visuals (e.g., precision-recall curve.

On the other hand, ML testing involves checks on model behaviour. Pre-train tests—which can be run without trained parameters



### 2.2.1 What does machine models benefit from testing?

- **Adversarial attacks**: Testing models can help detect possible adversarial attacks. Rather than letting this attack happen in a production environment, a model can be tested with adversarial examples to increase its robustness prior to deployment.
- **Data integrity and bias**: Data collected from most sources are usually unstructured and might reflect human bias that can be modeled during training. This bias might be against a particular group either by gender, race, religion, or sexuality with varying consequences in society depending on the scale of use. During the evaluation, bias can be missed because it focuses mostly on performance and not the behavior of the model given the role of the data in this case.
- **Spot failure modes**: Failure modes can occur when trying to deploy ML systems into production. These can be due to performance bias failures, robustness failures or model input failures. Some of these failures can be missed by evaluation metrics although they can signal problems. A model with an accuracy of 90% means that the model is finding it difficult to generalize with the 10% of the data. That can prompt you to check the data and look for errors giving you better insights on how to solve it. It is not all-encompassing and so structured tests for the possible scenarios that maybe encountered need to be established and help detect failure modes.

### 2.2.2 Problems with Testing Machine Learning Models

Software developers write codes to produce deterministic behavior. Testing identifies explicitly which part of the code fails and provides a relatively coherent coverage measure (e.g., lines of code covered). It helps us in two ways:

- Quality assurance; whether the software works according to requirements
- Identify defects and flaws during development and in production.

Data scientists and ML engineers train models by feeding them examples and setting parameters. The model's training logic produces the behavior. This process poses these challenges when testing ML models:

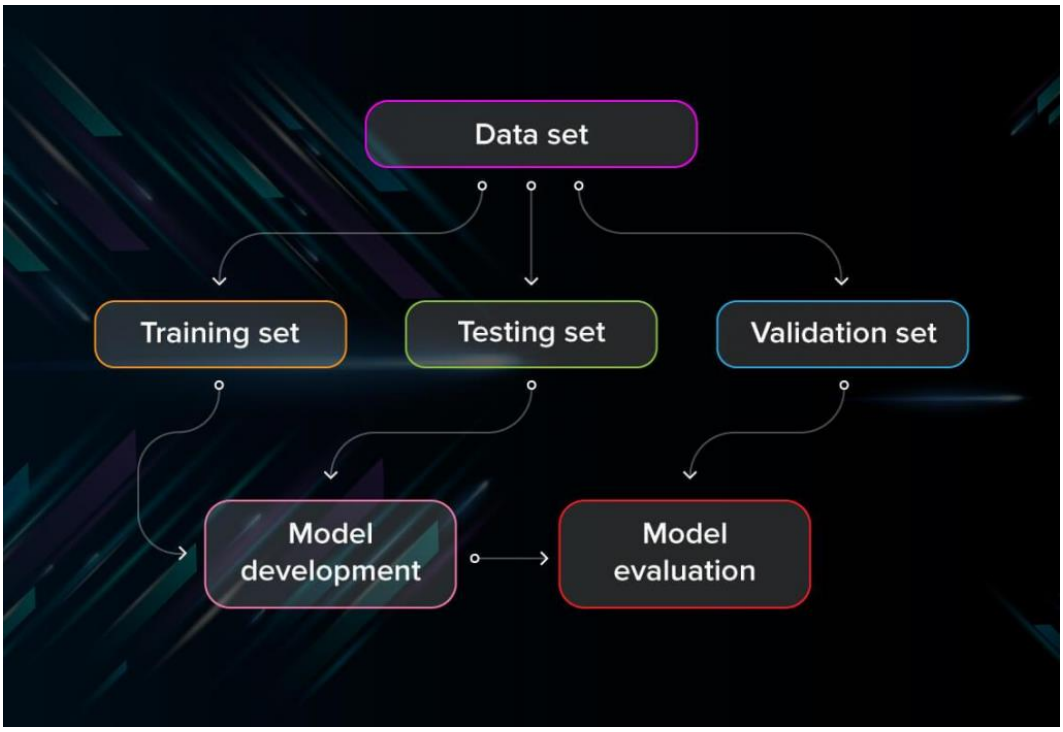
- Lack of transparency. Many models work like black boxes.
- Indeterminate modeling outcomes. Many models rely on stochastic algorithms and do not produce the same model after (re)training.

- Generalizability. Models need to work consistently in circumstances other than their training environment.
- Unclear idea of coverage. There is no established way to express testing coverage for machine learning models. “Coverage” does not refer to lines of code in machine learning as it does in software development. Instead, it might relate to ideas like input data and model output distribution.
- Resource needs. Continuous testing of ML models requires resource and is time-intensive.

These issues make it difficult to understand the reasons behind a model’s low performance, interpret the results, and assure that our model will work even when there is a change in the input data distribution (data drift) or in the relationship between our input and output variables (concept drift).

### 2.2.3 Evaluation and Testing

ML model evaluation focuses on the overall performance of the model. Such evaluations may consist of performance metrics and curves, and perhaps examples of incorrect predictions. This model evaluation is a great way to monitor your model’s outcome between different versions. Remember that it does not tell us a lot about the reasons behind the failures and the specific model behaviors



Machine learning tests, on the other hand, go beyond evaluating the models' performance on subsets of data. It ensures that the composite parts of the ML system are working effectively to achieve the desired level of quality results. It helps teams point out flaws in the code, data, and model so they can be fixed.

### 2.2.4 Principles and best practices

- Test after introducing a new component, model, or data, and after model retraining.
- Test before deployment and production.
- Write tests to avoid recognized bugs in the future.

Testing ML models has additional requirements. You also need to follow testing principles specific to the ML problem:

- Robustness
- Interpretability
- Reproducibility

### 2.2.5 How to Test Machine Learning Models

- Unit test. Check the correctness of individual model components.
- Regression test. Check whether your model breaks and test for previously encountered bugs.
- Integration test. Check whether the different components work with each other within your machine learning pipeline.

### 2.2.6 Testing trained models

For code, you can write manual test cases. This is not a great option for Machine Learning models as you cannot cover all edge cases in a multi-dimensional input space.

Instead, test model performance by doing monitoring, data slicing, or property-based testing targeted at real world problems.

You can combine this with test types that examine specifically the internal behavior of your trained models (post-train tests):

- Invariance test
- Directional expectation test
- Minimum functionality test

**Invariance test**
The invariance test defines input changes that are expected to leave model outputs unaffected
The common method for testing invariance is related to data augmentation. You pair up modified and unmodified input examples and see how much this affects the model output.
**Directional Expectation Test**
You can run directional expectation tests to define input distribution changes expected effects on the output.

The minimum functionality test helps you decide whether individual model components behave as you expect. The reasoning behind these tests is that overall, output-based performance can conceal critical upcoming issues in your model.

Here are ways to test individual components:

- Create samples that are “very easy” for the model to predict, in order to see if they consistently deliver these types of predictions.
- Test data segments and subsets that meet a specific criteria (e.g., run your language model only on short sentences of your data to see its ability to “predict short sentences”).
- Test for failure modes you have identified during manual error analysis.

# REFERENCE

**English**

**[1]** https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

**[2]** https://www.datacamp.com/blog/what-is-continuous-learning

**[3]** https://applyingml.com/resources/testing-ml/

**[4]** https://deepchecks.com/how-to-test-machine-learning-models/