International Journal of Scientific Research and Engineering Development-– Volume X Issue X, Year

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Strategies for effective ML testing

Anupama Singh
(Amity school of engineering and technology, Amity University, Noida, India)
Email: anupamaasingh166@gmail.com

----------**********----------

Abstract:

In machine learning, a programmer usually inputs the data and the desired behaviour, and the logic is elaborated by the machine. This is especially true for deep learning. Therefore, the purpose of machine learning testing is, first of all, to ensure that this learned logic will remain consistent, no matter how many times we call the program.

Keywords — **Machine learning, deep learning, datasets.**

----------**********----------

I. INTRODUCTION

All models rely on data, but data does not exist in a vacuum. Talented professionals oversee managing pipelines, data warehouses, and data lakes. Even so, data can lose its value. Specific data fields might go out of use or no longer be updated. Certain data-hygiene processes might no longer followed. Whatever the cause of the data degradation, models based on that data can be destined to failure. Before starting any data-based project, make sure you are fully aware of that data's quality, completeness, and update frequency, as well as of any anticipated changes to the database. The purpose of machine learning testing is to ensure that the software system meets the requirements, quality assurance is essential. Were all the agreed-upon features implemented? Is the programme performing as expected? The technical specification document should list all the parameters against which you will test the application. Furthermore, software testing could identify any problems and weaknesses during the development process. You don't want your customers to come to you swinging their fists after discovering defects in the product after it's been deployed. We can catch flaws that are only visible at runtime using various types of testing. In machine learning, however, the data and desired behaviour are normally entered by a programmer, and the logic is elaborated by the computer. This is particularly true in the case of deep learning. As a result, the goal of machine learning testing is to ensure that the taught logic remains consistent no matter how many times the programme is called.

II. MODEL EVALUATION IN MACHINE LEARNING TESTING

Usually, software testing includes:

- ***Unit tests***. The program is broken down into blocks, and each element (unit) is tested separately.
- ***Regression tests***. They cover already tested software to see if it doesn't suddenly break.
- ***Integration tests.*** This type of testing observes how multiple components of the program work together.

Furthermore, individuals follow specific rules: don't merge code until all tests pass, always test newly added blocks of code, and when resolving defects, develop a test that captures the bug. Machine learning increases the number of tasks on your to-do list. You must still adhere to ML's best practises. Furthermore, every machine learning model must not only be tested but also appraised. Your model should be able to be applied to a wide range of

situations. Although this is not what we generally think of when we think of testing, evaluation is necessary to ensure that the performance is sufficient.
First of all, you split the database into three non-overlapping sets. You use a training set to train the model. Then, to evaluate the performance of the model, you use two sets of data:

- *Validation set*. Having only a training set and a testing set is not enough if you do many rounds of hyperparameter-tuning (which is always). And that can result in overfitting. To avoid that, you can select a small validation data set to evaluate a model. Only after you get maximum accuracy on the validation set, you make the testing set come into the game.

- *Test set (or holdout set*). Your model might fit the training dataset perfectly well. But where are the guarantees that it will do equally well in real-life? In order to assure that, you select samples for a testing set from your training set — examples that the machine hasn't seen before. It is important to remain unbiased during selection and draw samples at random. Also, you should not use the same set many times to avoid training on your test data. Your test set should be large enough to provide statistically meaningful results and be representative of the data set as a whole.

Validation sets, like test sets, "wear out" with regular use. The more you utilise the same data to make decisions about hyperparameter settings or other model changes, the less confident you are in the model's ability to generalise successfully to new, unknown data. As a result, collecting extra data to 'freshen up' the test and validation sets is a smart idea.

Cross-validation

Cross-validation is a model evaluation technique that can be performed even on a limited dataset. The training set is divided into small subsets, and the model is trained and validated on each of these samples.

k-fold cross-validation

The most common cross-validation method is called k-fold cross-validation. To use it, you need to divide the dataset into k subsets (also called folds) and use them k times. For example, by breaking the dataset into 10 subsets, you will perform a 10-fold cross-validation. Each subset must be used as the validation set at least once. This method is beneficial for evaluating the machine learning model's performance on previously unseen data. It's so popular because it's simple to use, works well even with little datasets, and produces generally accurate results.

Leave-one-out cross-validation

In this method, we train the model on all the data samples in the set except for one data point that is used to test the model. By repeating this process iteratively, each time leaving a different data point as a testing set, you get to test the performance for all the data. The benefit of the method is low bias since all the data points are used. However, it also leads to higher variation in testing because we are testing the model against just one data point each time. Cross-validation allows for more efficient data consumption and aids in determining the model's correctness.

III. BEST PRACTICES FOR ML MODEL DEBUGGING

We still need to figure out where and why the errors occur once we've examined the performance. Debugging machine learning is a little different from debugging any other software system. An ML model's poor prediction quality does not always imply that it has a flaw. You must look into a wider range of problems than in traditional programming: perhaps the data contains mistakes, or the hyperparameters are not properly tuned. This makes debugging ML models extremely difficult.

Data debugging

First of all, you should start with data debugging because the accuracy of predictions made by the model depends not only on the algorithm but on the quality of data itself.

Database schema

One tool that helps you to check whether the data contains the expected statistical values is the data schema.

A database schema is like a map that describes the logic of the database: how the data is organized and what the relationship between the samples is.

It may include certain rules like:

- Ensure that the submitted values are within the 1-5 range (in ratings, for example).
- Check that all the images are in the JPEG format.

The scheme can be of two types:

- Physical.

It describes how the data will be stored, its formats, etc.

- Logical.

This type represents the logical components of the database in the form of tables, tags, or schemes.

Separate checks should be performed on the engineered data. While raw data may be acceptable, manufactured data has undergone modifications and may appear to be completely different. You may build tests to ensure that outliers are handled properly and that missing values are replaced with the mean or default values.

Model debugging

Once you have tested your data, you can proceed with model debugging. Establish a baseline When you set a baseline and compare your model against it, you quickly test the model's quality. Establishing a baseline means that you use a simple heuristic to predict the label. If your trained model performs worse than its baseline, you need to improve your model.

For example, if your model solves a classification problem, the baseline is predicting the most common label.

Once you validate your model and update it, you can use it as a baseline for newer versions of the model. An updated, more complex model must perform better than a less complex baseline.

Write ML unit tests

This kind of ML testing is more similar to traditional testing: you write and run tests checking the performance of the program. Applying the tests, you catch bugs in different components of the ML program. For example, you can test that the hidden layers in a neural network are configured correctly.

Adjust hyperparameters

Ill-adjusted hyperparameters can be the reason for the poor performance of the model.

Here are the metrics you should usually check:

- Learning rate.

Usually, ML libraries pre-set a learning rate, for example, in TensorFlow it is 0.05. However, it might not be the best learning rate for your model. So, the best option is to set it manually between 0.0001 and 1.0 and play with it, seeing what gives you the best loss without taking hours to train.

- Regularization.

You should conduct regularization only after you have made sure that the model can make predictions on the training data without regularization. L1 regularization is useful if you need to reduce your model's size. Apply L2 regularization if you prefer increased model stability. And, in the case of neural networks, work with dropout regularization.

- Batch size.

Models trained on smaller batches usually generalize better. A batch should usually contain 10-1000 samples where the minimal size depends on your model.

- Depth of layers.

The depth describes the number of layers in a neural network: the more layers it has, the deeper it is. Start from 1 layer and gradually increase the number if you feel like the model should be deeper to solve your problem. This approach helps to not overcomplicate the model from the very beginning.

IV. HOW TO WRITE MODEL TESTS

So, to write model tests, we need to cover several issues:

- Check the general logic of the model (not possible in the case of deep neural networks so go to the next step if working with a DL model).
- Control the model performance by manual testing for a random couple of data points.
- Evaluate the accuracy of the ML model.
- Make sure that the achieved loss is acceptable for your task.
- If you get reasonable results, jump to unit tests to check the model performance on the real data.

There are two general kinds of tests:

Pre-train tests

This type of test is performed early on and allows you to catch bugs before running the model. They do not need training parameters to be run. An example of a pre-train test is a program that checks whether there are any labels missing in your training and validation datasets.

Post-train tests

These tests are performed on a trained model and check whether it performs correctly. They allow us to investigate the logic behind the algorithm and see whether there are any bugs there. There are three types of tests that report the behavior of the program:

- Invariance tests.

Using invariance tests, we can check how much we can change the input without it affecting the performance of the model. We can pair up input examples and check for consistency in predictions. For example, if we run a pattern recognition model on two different photos of red apples, we expect that the result will not change much.

- Directional expectation tests.

Unlike invariance tests, directional expectation tests are needed to check how perturbations in input will change the behavior of the model. For example, when building a regression model that estimates the prices of houses and takes square meters as one of the parameters, we want to see that adding extra space makes the price go up.

- Minimum functionality tests.

These tests enable us to test the components of the program separately just like traditional unit tests. For example, you can assess the model on specific cases found in your data.

V. CONCLUSION

If you're concerned about the model's quality, you'll need to run ML tests. ML testing has a few quirks: it requires you to evaluate the quality of the data rather than simply the model, and it requires you to go through several rounds tweaking the hyperparameters to achieve the best results. However, if you follow all of the essential steps, you may rest assured that it will work.

REFERENCES

[1] Michael Chu - Data Science Projects – Medium For important decisions, listen to your AI, but retain responsibility
[2] Testing Machine Learning Models - Knowledia News
[3] Testing ML: A Step Towards Perfection – Anteelo
[4] Descending into ML: Training and Loss- Google Developers.com
[5] Training ML Models - Amazon Machine Learning

