Software Testing for Machine Learning

A survey of software testing methods used in machine learning

Hugo Prevoteau (悠够)
6219000083
College of Intelligence and Computing
Tianjin University, China
prevoteauhugo@gmail.com

INTRODUCTION

Machine learning is a sub-section of computer programming using probabilities and statistics to allow computers to learn by themselves without being explicitly brought to it.

The main objective is to teach them how to learn, so that later computers can take decisions on their own which relies on the large amount of data fed to the algorithms.

Today, machine learning is widely used: it allows advertising companies to target better products to advertise to each individual, Tesla cars to drive by themselves, or even some video surveillance algorithms.

Machine learning is declined into three categories, supervised (Classification, Regression), unsupervised (Clustering, Association), reinforcement (Monte Carlo, Temporal Difference).

As these methods are being adopted by many areas, their reliability and robustness are becoming very important. We then needed to develop techniques to test machine learning algorithms.

Software testing is used to find bugs in code, as each program produces an output to an input, testing a program means checking that the code replies correctly to any input. Software testing is declined in many test types, that all have a particular purpose and are therefore complementary. The testing methods will identify errors, faults and anomalies on the tested application.

Although machine learning is not tested the same way as standard algorithms, the goal is the same: to check that the output is always the expected one, and that the solution works under any conditions.

Remark: as the most complex and testing worth models are deep learning models, most of the applications and examples given is this report will be related to this variant of machine learning.

1 Standard testing methods

This section discusses the main testing methods used on machine learning solutions.

1.1 Test oracles

1.1.*1* Definition

The term “test oracle” indicates the correct output to an arbitrary input. It is the rule set by the testers to define if a test has past or failed. In the facts, we will compare the real output to the output generated respecting the test oracle rules.

The test oracles can be defined by many ways, through statistics, program documentation or even be model based. Factually it is a real challenge to define the test oracles in machine learning as the expected outputs may not be precisely defined in some cases, and the logic of the algorithm is too complex to be manually reproduced (i.e. neural networks).

1.1.*2* Generating test oracles

1.1.2.1 Multiple implementations

In most cases, to generate the test oracles, we use different implementations of the same algorithm to compare their outputs resulting from the same input. Then, the result acknowledged by most of the implementations is considered as the correct output regarding this precise input. This method requires attention over the independence of the different implementations.

1.1.2.2 Metamorphic testing

First, some program relative properties need to be identified, they are called metamorphic relationships. They denote the relations between a group of inputs and their corresponding outputs. After that, we test these relationships over other cases, if they do not hold, then the program may have defect that requiring a fix.
Factually it means that two similar outputs should be given the same label by the algorithm.
This method is very effective but requires to manually identify the previously defined relationships.

1.2 Test coverage

1.2.*1* Definition

Test coverage is a measure used to define the degree of which the program is accessed through the test inputs. Which means that the higher this test coverage is, the lower the number of undetected bugs is.

As a matter of fact, machine learning requires much less different test methods to reach a satisfying test coverage.

1.2.2 Approach

Because of the differences between machine learning and traditional applications, some metrics were proposed, such as the neuron coverage, what results from the number of activated neurons divided by the number of total neurons. This metric shows how much of the learned knowledge is assessed through this iteration.

1.3 Corner cases

1.3.*1* Definition

As a machine learning is trained on a specific dataset that can be biased, it is important to make sure that the algorithm will correctly address every input it is given. This is the reason why it is important to generate corner cases.

1.2.2 Approach

Three main approaches exist. The first one consists of changing the value of the source case of the gradient ascent. The second one consists of transforming the application scenarios, and the last one is generating adversarial examples. It is also important to mention that the architecture of the corner cases generation and testing will differ a lot through machine learning applications.

For example, for image classification, altering the input pictures by adding noise, deformation or changing the colors may be a good way to test if the model still performs well. One of the major fixes to these bugs is to use “data augmentation” techniques that will reduce overfitting and then increase the corner cases coverage of the model.

1.4 Parameter Testing

As the execution of a machine learning algorithm depends a lot on its parameters, to fully test a machine learning implementation we need to test all the possible parameters.

This task is specifically complex for machine learning algorithms as most of the time the number of parameters is very high, and the number of possible combinations quickly becomes humongous. In practice, it is impossible to cover all the parameters.

2 Testing techniques used in machine learning

This part will discuss the testing techniques discussed in the module that are used to test machine learning models. The techniques are fuzzy testing, regression testing, mutation testing, concurrent testing, load testing and model checking.

2.1 Fuzzing testing

2.1.1 Definition

This testing technique is used to discover bugs occurring only for rare input. For neural networks, coverage-guided fuzzing is used to find serious bugs, by applying random mutations to the initial input set according to defined mutation procedures.

Coverage can be increased by mutating the data and see the activation sequence of the different neurons. If a new sequence is detected: the coverage has increased.

Fuzzing is used to see if mutated data activates the same neurons as the initial input and the outputs can also be compared.

2.1.2 Parameters choice

There are three steps to follow to perform a fuzzing on a dataset.

1. Input chooser: at any given time in the testing process the fuzzy tester needs to define which input to mutate, to take this decision a set of heuristics is used.
   This choice may highly change the execution speed and the performance of the testing.
2. Mutator: define the mutations that we want to apply to our inputs. It fully depends on the dataset we are using, if it is a graph dataset, we may mutate node links or edge labels, if it is pictures, we can change the size, or crop it.
3. Coverage analysis: defining wether the coverage increased or not by looking at the different metrics provided by the method (for neural networks it could be the neuron activation sequence)

After that naturally comes the step where we check that the outputs match the expected outputs, so the model contains no bugs.

2.3 Mutation testing

2.3.1 Definition

To perform a mutation testing there is a process to follow, given a program *P*, a set of faulty programs *P'* is created based on defined rules, where each of them slightly mutates *P*. Then we use a set of tests *T* on *P*. The passed test, a subset of *T* is called *T'* and applied to the different mutations of the program noted *P'*. If the results of a mutant program of *P'* on *T'* is different than the one from *P,* then the mutant program is killed. After all the *T'* tests have been processed we can compute the mutation score (*number of killed mutants/number of mutants*). After the mutation testing the developers gets some information over the quality of his test and can improve them or add some more.

For machine learning, we fill mutate the initial program *P* and the initial dataset *D* creating a *D'* for each *P'*.
In fact, the process for mutation testing is always the same, what differs from an application field to another is the mutation operators.

2.1.2 Mutation operators
There are two distinct groups of mutators to differentiate, first data mutation operators, and program mutation operators.

2.1.2.1 Data mutation operators
Some data mutation operators:
1. Data repetition: some parts of the dataset are duplicated
2. Label error: injecting wrongly labeled inputs in *D'*
3. Data missing removes some of the training data
4. Data shuffle: shuffles the data before the training starts
5. Noise perturbation: randomly adds noise to the training data

2.1.2.2 Program mutation operators
Some program mutation operators:
1. Layer removal: randomly removes a layer from the model
2. Layer addition: randomly adds a layer to the model
3. Activation function removal: mimic the situation where the developer forgets to add the activation functions by removing all the activation functions of a layer

2.4 Load testing

2.4.1 Definition
To test if a software is stress-resistant, load testing procedures are applied. These procedures will compare the system behavior under both normal and peak load conditions. The goal is to check if the model is working normally under stress. Whether it is about determining the maximum operating capacity of an application or highlighting inefficient procedures, load testing is a big step of software testing specially in machine learning where the efficiency of the model is crucial.
Factually, a machine learning algorithm could perform very well on a small dataset and react differently when computed on the cloud with much larger datasets.

2.4.2 Testing an algorithm
Indeed, increasing the size of the dataset, trying the algorithm under real life-stress. In practice before being deployed all solutions are tested in a pre-production environment, so some execution tests can be fixed before deploying it.
Checking the time required to make a prediction or to go through the dataset is measured during the test and decisions are made based on if the reactions are normal or abnormal.
Also, it is important to mention that very small bugs in the code can result in very bad execution time which can be crucial in some cases (i.e. High Frequency Trading).

3 Additional methods

This section will debate methods that are not (or not widely) used for testing on machine learning algorithms. It will explain how these methods would work on my research work particularly.
To introduce my research work more precisely; I am working on the analysis of circulatory behaviors using graph theory. The range of analysis I am carrying out is very large and applying machine learning on graphs has been one of the main contributions.
While many machine learning algorithms have been published, only a few portions of them are available in python APIs such as there would be for images or text with TensorFlow or Keras.
So, I have been brought to improve some methods, and to make sure I am truly improving them, I tested many possible cases that would crash by algorithm.

3.1 Regression testing

2.2.1 Definition
Regression testing is a method used to measure the impact of the new versions in terms of bugs. In point of fact it is mostly used in two cases: the first one is when a software is updated, we need to check that the added functionalities did not bring any bug in the program; the second reason is that while correcting bugs we may create some other ones, so we need to be careful about this also.
Factually that

2.2.2 Possible use cases
For my research project, I am computing the Maximum Spanning Tree using Prim's algorithm over a dataset of graphs. If I want to use a graph embedding deep learning method, I have first to verify it will work properly on the case that trees are.
What is interesting here is if the algorithm does not support tree embedding. I would need to modify the source code so the embedding works. But other difficulties arise, such as how can I be sure that the embeddings are now working as well as they were on regular graphs, will they still work on directed acrylic graphs, etc.
So, here is a very interesting case where regression testing is important and could be used: while editing the source code of the method so it can deal with a new type of data, am I not creating more bugs than I am fixing.

4 Additional methods not described in the module

This part will talk about techniques that are used to test machine learning software, with techniques that are not described in the class. It means that these techniques are applied to machine learning specifically.

4.1 Validation testing

4.1.1 Motivation

This method is exclusively used to test machine learning algorithms, particularly to make sure the model works correctly. In fact, when implementing a machine learning algorithm, we need to make sure that the model does not overfit or underfit. It means that we are not over learning on the data or under learning. The best-case scenario is when we are just learning, which means that there are not too many iterations on the dataset, each case is well represented.

4.1.2 Definition

In practice, to validate a machine learning model, we split the training set in two distinct parts. We will learn on the first one and try to predict the second one. As both are part of the training set (which means that we know the real answer) we can compare the prediction and the actual label.

4.1.3 Use case

In most deep learning method, there is many parameters that we can adjust, such as the learning rate for example which is the fraction of the hyperparameters that we will update at each iteration (if this value is too low it takes too much time to learn out of the data, so the gradient descent can not happen on the given iterations, and if this value is too high we may jump over the value we want to reach.) and we need to make sure the learning is coherent. So, we use testing extensions such as TensorBoard to visualize the accuracy evolution through iterations. We can then detect overfitting or underfitting.

CONCLUSION

To conclude over software testing on machine learning algorithms we can identify that indeed many tests are processed, but the difficulty to define a test oracle makes machine learning testing difficult and laborious, yet no proper framework has been proposed. While many research papers talk about testing on machine learning, it remains a largely experimental field, such as machine learning itself.

It is worth to mention that machine learning itself is used to test other software as well.

REFERENCES

[1] Song Huang, 2018. Challenges of Testing Machine Learning Applications DOI: 10.23940/ijpe.18.06. p18.12751282
[2] Moving targets: testing software in the age of machine learning Website: techbeacon.com
[3] Qu'est-ce que le machine learning? ('What's machine learning?') Website: fr.talend.com
[4] Augustus Odena, 2018. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing DOI: arXiv:1807.10875
[5] Christian Murphy, 2007. An approach to Software Testing of Machine Learning Applications DOI: https://doi.org/10.7916/D8R49ZNR
[6] Paul Bruce, 2019. The importance of performance testing machine learning models. Website: neotys.com