TESTING MACHINE LEARNING ALGORITHMS WITHOUT ORACLE

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfilment
of the Requirements for the Degree

Master of Science in Computer Science

University of Nebraska at Omaha

by

Abhishek Kumar

August, 2018

Supervisory Committee:

Harvey Siy, Ph.D.

Myoungkyu Song, Ph.D.

Matthew Hale, Ph.D.

TESTING MACHINE LEARNING ALGORITHMS WITHOUT ORACLE

Abhishek Kumar, M. S.

University of Nebraska, 2018

Advisor: Harvey Siy, Ph.D.

Abstract here

ACKNOWLEDGMENTS

Acknowledgments here

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning algorithms are becoming increasingly popular and is already being applied in different sectors like: healthcare, finance, retail, etc.. This rising popularity has created a demand for more complex and sophisticated algorithms in order to implement state-of-the-art use cases. While developing and training an implementation of a machine learning model the aim is to create a model that makes the best predictions. Lack of reliable oracles for testing machine learning algorithms makes it very hard to test the accuracy of such implementations. Such set of programs which does not have a test oracle that can predict the output on a set of inputs are called "non-testable programs"[**?**]. Davis and Weyuker describe these set of programs as "Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known". Most machine learning programs fall under this category as they are written in order to predict the correct answers in the first place. In this paper we will explore the types of guarantees one can expect a machine learning model to possess based on the properties that the underlying algorithm of the implementation possess.

In order to design reliable systems, engineers typically engage in both testing and verification:

- By testing, we mean evaluating the system in several conditions and observing its behavior, watching for defects
- By verification, we mean producing a compelling argument that the system will not misbehave under a very broad range of circumstances.

Dataset coverages and We propose applying MT as a way of testing and verifying ML programs.

Chapter 2

Literature Review

2.1 Testing Without Oracles

The current testing research activities fall under three categories: Developing a sound theoretical basis for testing. Devising and improving testing methodologies, especially the mechanizable ones. Defining accurate measurement criteria for testing data adequately. An oracle is a system that determines the correctness of the solution by comparing the systems output to the one that it predicts. A program is considered non-testable if one of the following two conditions occur: A oracle does not exist for the given problem. It is theoretically possible to determine the correct output but computationally very hard. The programs that dont have oracles can be usually classified in three categories: Programs that were written to determine the correct answer. Programs that produce lot of outputs such that it is hard to verify all of them. Programs where tester have a misconceptions (tester believes that he has the oracle even though he might not). Pseudo Oracles/Dual Coding: Another set of program is written independently according to the same specification as the original program and the output from both the programs is compared. If the outputs match it can be asserted that the original results are according to the specification. The problem with dual coding is that it has a lot of overhead and requires more time and money. This is done only for highly critical softwares. While performing mathematical computations, errors from three sources can creep in: The mathematical model used to do the computations. Programs written to implement the computation. The features of the environment like: round-off, floating point operations etc. Even in the absence of oracles the users often have a ballpark idea of what the correct answer would look like without knowing the correct answer. In such cases we make use of partial oracles. It is relatively easier to test the systems on simpler inputs for which the output is known. The problem, of course, is that from experience we know that most errors occur in complicated test-cases. It is common for central test cases to work and boundary cases to fail. From the above observations the authors make five recommendation for items to be considered as a part of documentation. The criteria used to select the test data. The degree to which the criteria was fulfilled. The test data, the program ran on. The output of each of each test datum. How the results were determined to be correct or acceptable. Although the recommendations do not solve the problem of non-testable programs but they do provide information on whether the program should be considered adequately tested or not.

2.2 Metamorphic Testing

2.2.1 Automatic System Testing of Programs without Test Oracles

In this paper the authors have demonstrated the usefulness of metamorphic testing in assessing the quality of applications without test oracles. Comparing the outputs of the morphed data still remains a challenge especially if the data set is large or not in human readable format. The authors presented an approach called “Automated Metamorphic System Testing” to automate the metamorphic testing by considering the system as a blackbox and checking if the metamorphic properties holds after execution of the system. They also present another approach Heuristic Metamorphic Testing to reduce false positives and address some non-determinism. Unlike in the previous papers, here the authors are focusing to improve the metamorphic testing technique itself. They list some benefits of using metamorphic testing: it can be used on broader domain of applications that display metamorphic properties, and it treats the application under test as a black box and does not require detailed understanding of the source code. They then list some of the limitations of using metamorphic testing: Manual transformation of large input data can be laborious and error-prone. They need special tools to transform the input. Comparing the outputs(some of which may be very large and/or in not human-readable format) of the input data can be tedious. Floating point calculations can also lead to imprecision even though the calculations are programmatically correct. Coming up with the initial test-cases is also a challenge as some defects may only occur under certain inputs. Automated Metamorphic System Testing: This technique can be used to test the application in development environment as well as in production as long as the users are only provided the output from the original execution and not the result from transformed input. In this model: Metamorphic properties are specified by the tester and applied to the input. The original input is fed into the application which is treated as a blackbox and a transformation of the input is also generated. That transformed input is fed into a separate instance of the application running in a separate sandbox. When the invocations are finished, the results are compared and if they do not match according to the specifications, there is an error. Tester need not write any code and only needs to specify the metamorphic properties. They dont need to know the source code or other implementation details. Amsterdam framework: The metamorphic properties are specified using XML file. The specification consist of three parts: how to transform the input, how to execute the program, and how to compare the outputs. Heuristic Metamorphic testing: This method allows for small differences in outputs, in a meaningful way according to the application being used to address the problems of false positives and non-determinism. Imprecisions in floating point calculation and representation of irrational number such as may result in failure of metamorphic testing even if the implementation is correct. If two outputs are close enough they are considered the same. The definition of close enough depends on the application and in complex applications checking semantic similarity may also be required.

2.3 Overview of Machine Learning Algorithms

2.4 Testing Machine Learning Programs

2.4.1 Properties of Machine Learning Applications for Use in Metamorphic Testing

In the absence of a reliable oracle to indicate the correct output for arbitrary inputs, machine learning programs are often very hard to test. The general term for such softwares that do not have a reliable test oracle is non-testable programs. Such programs can be tested in one of the two ways:

- Creating multiple implementations of the same program and testing them on same inputs and comparing the results. If the outputs are not same then either of the implementations can contain error. This approach is called pseudo-oracle.

- In absence of multiple oracle, metamorphic testing can be used. In metamorphic testing, the input is modified using a metamorphic relation such that the two sets of input will generate similar outputs. If similar outputs are not observed then there must be a defect.

The main challenge with metamorphic testing is to come up with the metamorphic relations to transform inputs since such coming up with such relations require domain knowledge and/or familiarity with the implementation. In this paper the authors seek to create a taxonomy of metamorphic relationships that can be applied to the input data for both supervised and unsupervised machine learning softwares. These set of properties can be applied to define the metamorphic relationships so that metamorphic testing can be used as a general testing method for machine learning applications. The problem with some of the current machine learning frameworks like: Weka and Orange is that they compare the quality of results but dont evaluate the correctness of the results. The authors apply metamorphic testing to three ML applications: MartiRank, SVM-Light, PAYL. MartiRank is a supervised ML algorithm that applies segmentation and sorting of the input data to create a model. The algorithm then performs similar operations from the model on the test data to produce a ranking list. SVM-Light is an open-source implementation of SVM that also has a ranking mode. The authors also investigated an intrusion detection system called PAYL. PAYL is an unsupervised machine learning system. Its dataset simply consist of TCP/IP network payloads(stream of bytes) without any label or classification. Based on the analysis of MartiRank algorithm, the authors realized that the actual values of the attributes were not very important but their relative values determined the model. Thus, adding a constant value to every attribute or multiplying each attribute with a positive number, should not affect the model and generate the same ranking as before. Thus, the metamorphic properties identified were: addition and multiplication. Applying the model on two sets of data, one of which created from the other, either by multiplying a positive number or, adding a constant number, should not change the ranking. Changing the order of examples should not affect the model or ranking since the algorithm sorts the inputs thus, MartiRank also has permutative metamorphic property. Multiplying the data by a negative constant value will create a new sorting order which can easily predicted. The only change to the model will be the sorting direction i.e. the algorithm will change the sorting direction but keep the sorting order intact. Thus, MartiRank also displays an invertive metamorphic property where the output can be predicted by taking the opposite of input. MartiRank also includes inclusive and exclusive metamorphic properties. Knowing the model can help predict the position of any new elements.

2.4.2 Application of Metamorphic Testing to Supervised Classifiers

Building on the previous paper, the authors explore the metamorphic relations based on expected behavior of given machine learning problems. They present a case study on Weka, a popular machine learning framework, which is also the foundation for computational science tools such as BioWeka in bioinformatics. In this paper the authors explore k-Nearest Neighbors and Naive Bayes classifier algorithms. Previously, they researched on Support Vector Machines. In this paper the authors seek to identify the metamorphic relations for the two algorithms (kNN and NBC). NBC and kNN both calculates the mean and standard deviation of the input data. Thus, the metamorphic relations identified are: Permuting the order of input data does not affect the mean or standard deviation. Multiplying the data with -1 does not affect the standard deviation since, the deviation from the mean will still be the same. Multiplying the data with some other positive number will increase the standard deviation by the same amount. Thus, the output will still be predictable. The authors then, define the metamorphic relations that a classification algorithm is expected to exhibit:

1. Consistency with affine transformation.

2. Permutation of class labels.

3. Permutation of attributes.

4. Addition of uninformative/informative attributes.

5. Consistency with re-prediction.

6. Addition of training samples.

7. Addition of classes by duplicating/re-labeling samples.

8. Removal of classes/samples.

Next, the authors introduce the notion of validation and verification. Validation refers to choosing the most appropriate algorithm to solve a problem. Verification refers to whether the implemented algorithm is correct or not. Current, software testing methods have not addressed the problem of validation and only focus on verification. The authors then performed an experiment to verify the correctness of Weka. They created a set of random input data and used the above metamorphic relationships to generate another set of inputs. Upon running the inputs on both the algorithms they realized only a subset of MRs were a necessary property of the corresponding algorithm. It was observed that several MRs violated NBC algorithms. Violations in the MRs that are necessary properties imply defects in implementation. In the case of kNN algorithm, none of the necessary MRs were violated which means that there are no implementation error as per the testing.

2.4.3 Dataset Coverage for Testing Machine Learning Computer Programs

Recently, computer programs for Big Data analytics or statistical machine learning have become essential components of intelligent software systems. Test oracles are rarely available for them, and this unavailability of test oracles is known as the oracle problem. Machine learning programs are a typical instance of non-testable programs, and is of the known unknowns type. Metamorphic testing (MT) is a method for tackling the oracle problem. Metamorphic relations (MR) play a role as pseudo oracles to check whether executions of the same program differ for two different test inputs. The test inputs are related by translation functions derived from metamorphic properties so that the relationship between the two results is predictable. If the results coincide with each other, the program behavior is relatively correct. This paper studies the characteristics of the SUT, the supervised learning classifiers. Identifying Quasi-testable Core: A program component, function or procedure, is quasi-testable if we have appropriate pseudo oracles or metamorphic relations. The result of the program execution embodies uncertainty, because the output is accompanied with the statistical classification performance. The classifier itself is non-testable. However, pseudo oracles with a MT can be used for testing. Dataset Coverage: Test coverage is essential in software testing because it is a basis to measure how much of the SUT is checked with a set of the input test data. The graph coverage is the most popular model for software testing, because it captures the structural characteristics of software artifacts, such as control-flows or data-flows of a computer program. The paper introduces the notion of dataset coverage to focus on the characteristics of the population distribution in the training dataset. However, complete coverage is not possible. The number of possible populations in datasets is also infinite. SVM: A support vector machine (SVM) is a supervised machine learning classifier. The support vectors lie on the dotted hyperplanes parallel to the separating hyperplane. The margin, the minimum gap between the support hyperplane and the separating hyperplane, is chosen to be maximum. The pseudo code is a common, abstract description of implemented SMO computer programs. Because SMO is an algorithm for solving the SVM optimization problem, it corresponds to the model constructor and is an abstract version of the quasi-testable core.

Testing SVMs Main tasks:

- Obtain pseudo oracles.
    - MR for Pseudo Oracles: Various combinations of dataset is obtained by reordering the dataset.
    - MR for Dataset Generation: Dataset is increased to increase the population of the input dataset.
- Generate data points that achieve the required dataset coverage: In order that the result is predictable, the population distribution of the initial dataset is simple enough to contain linearly separable data points. Then a series of tests with pseudo oracles that are obtained based on appropriate metamorphic properties is conducted. Then dataset is extended by adding new data points to calculate a new hyperplane.

Similar metamorphic testing approach can be applied to K-nearest neighbors and a naive Bayes classifiers. Since testing the whole program at once is not always possible choosing a right SUT from a non-testable program has a large impact on testing activities.

Chapter 3

Proposed Work

3.1 Setting up the Test Environment

3.1.1 Docker

For replication of results. Image can be downloaded from dockerhub. Attached volume for persisting data.

3.1.2 Jupyter

The Jupyter Notebook is an open-source web application that supports data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, etc. by allowing us to create and share documents that contain live code, equations, visualizations and narrative text. Jupyter allows for displaying the result of computation inline in the form of rich media (SVG, LaTeX, etc.). This allows us to

3.1.3 Tensorflow

TensorFlow$^{\text{TM}}$ is an open source software library developed within Googles AI organization by the Google Brain team with a strong support for machine learning and deep learning. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. It is being used at a number of well known companies like Uber, Google, AMD, etc. for high performance numerical computation and machine learning. While TensorFlow is capable of handling a wide range of tasks, it is mainly designed for deep neural network models.

3.1.4 MNIST Dataset

The MNIST database of handwritten digits, acquired from http://yann.lecun.com/exdb/mnist/, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

3.1.4.1 Format of Dataset

3.2 Selection of Implementations to Test

Various models from TensorFlow is being used at different organizations like Mozilla, Google, Stanford University, etc. in different domains extending from speech recognition to computer vision. To evaluate the accuracy of some of the popular algorithms used in supervised classification we decided to implement metamorphic testing of:

- K-Nearest Neighbors
- SVM
- Neural Networks

Chapter 4

Work Plan