Mapping Definitions

Alexander Simola Bergsten,
Erik Bock

April 7, 2022

This document outlines the different tags used in the guidance mapping. The title of each section corresponds to a column name in the mapping. Definitions are provided for all the tags that can be used in each field/column of the mapping. These tags represent different testing goals, components under test, ML tasks, access levels, offline/online testing, testing workflows, and testing techniques as well as some limitations.

Goal

Adversarial Robustness
The model should be resistant to adversarial attacks, i.e., certain input that is tailored to exploit weaknesses in a model. Basically, it should be tough to fool the model [1]. The component has known weaknesses that we want to remedy, so the testing here exploits these weaknesses (often with tailored input targeted at the weaknesses).

Noise Robustness
The model can handle noisy input accurately. Compared to adversarial robustness, this goal is more about being able to handle out-of-sample inputs and data that is noisy, as opposed to resolving weaknesses of a model that can be exploited by hackers. Terms like consistency and stability are equivalent to this goal. Helps with issues such as data drift and, as the name implies, data that is prone to containing noise.

Group Fairness
Groups of individuals that differ only by some protected attribute[1] should have the same probability of a decision outcome [2]. If we divide a population on only a protected attribute, then we expect these segments to have a fairly similar distribution of output predictions. For example, if the model are classifying an input as "yes" or "no", then the segments should have a close to the same number of predicted "yes"/"no".

Individual Fairness
Two individuals that share many of the same traits but differ in some protected attribute[1] should be treated similarly, i.e., they should get the same decision output from the ML-based system [3]. What counts as a protected attribute depends on the domain, but for example, two otherwise very similar persons should not be treated differently purely by gender (in most cases).

Counterfactual Fairness
The model output for a specific individual should not change if the value of a protected attribute[1] is switched [4]. For a system input corresponding to a single individual, the system output should not differ if the same input is altered only in a protected attribute. It is a more extreme case of Individual fairness, where instead of only thinking of similar individuals the focus is on identical

[1] A *protected attribute* refers to a trait that should be protected from discrimination, for example gender, income, ethnicity.

individuals.

Interpretability
Is based on the definition in [2]. The goal of interpretability is that the models should be able to be understood by humans. The higher the interpretability the easier it should be to understand the model. For example, a decision tree is generally more interpretable than a neural network. Interpretability can be defined partly as transparency, i.e. the mechanism of the model is understandable, and partly by the concept of post hoc explanations, e.g. the decisions made by a MLS can be explained clearly.

Safety
Is based on the ISO 25010 definition [5] of *Health and Safety Risk Mitigation*. This goal essentially means that there is a high focus on ensuring that failures will not occur since this can lead to loss of time, resources, or in worst case human life. Software where this goals is in focus is often referred to as safety-critical software.

Privacy
Privacy has been defined in this guidance mapping with regards to how data, that can be tied to a single individual, is stored and used. To have a high amount of privacy the data needs to be stored securely, and also that it is anonymised if possible.

Security
Is based on the ISO 25010 definition [5] of *Security*. Essentially this goal means that controlling access-levels for different types of users is important. For an ML model, for example, a normal user might only be able to see predictions while a developer would also be able to modify the inputs that generated the prediction. Together with Adversarial Robustness, this goal can help keep malevolent agents from abusing the models or machine learning systems.

Correctness Is based on the ISO 25010 definition [5] of *Functional Correctness*. The term is used differently in many papers recommended by the mapping, but the goal is essentially that the model should be within some bound of precision for the tasks that it has been created for. The choice was also made to include papers focused on bug fixing in this goal, since the authors agreed that fixing bugs is meant to assure correctness of the model. In a specification, Correctness can be determined separately for each functional requirement. This means that we can have high Correctness for some, but low for others.

Completeness
Is based on the ISO 25010 definition [5] of *Functional Completeness*. Completeness is a goal that is related to correctness, but instead of focusing on high correctness for a requirement we look at how many of the requirements have been covered. This means that to achieve high Completeness we do not necessarily need high correctness for each requirement (although there still has to be an acceptable level).

Monotonicity
Based on the definition of the term used by Sharma and Wehrheim [6]. Essentially, having monotonicity means that variables that are directly correlated in the input and the output should always have this correlation when a prediction is made. If an input has a positive correlation to the output then an increase to the input value should always lead to an increase of the predicted output, given that

no other input values change as well.

Predictive Performance
A collected name for all the metrics that are commonly brought up when testing the performance of a ML model, such as Accuracy, Recall, Precision, F1-score, Specificity etc. This tool does not have a focus on this goal, although it is included as a secondary goal in some papers.

Efficiency
Is based on the ISO 25010 definition [5] of *Performance Efficiency*, which is essentially about how well a system performs compared to the resources used. In the papers tagged with this goal the focus is mostly to increase efficiency of training or testing by optimising test cases and input data.

CUT

Based on both Zhang et al. [2] and Riccio et al. [7], here we define the different components under test. Together with the Workflow tags, this tag is meant to help guide the developer to where in the pipeline the technique is meant to be implemented.

Data
Indicates that the testing is being done on the input data to the model. The most common type of testing here is checking if the data has any bias or invalid feature values.

Learning Program
Indicates that the testing is being done on the code that will create the model. This can be the ML algorithm itself, the training code, optimisation function, loss function etc.

Framework
Indicates that the testing is done on the framework that the model is created by. This can be testing built-in functions to make sure they behave as expected. For example PyTorch, TensorFlow, Scikit-learn.

Model
Indicates that the testing is conducted on a stand-alone model. For example a Neural Network (NN) or Support Vector Machines (SVM).

MLS
(i.e., Machine Learning-based System) Indicates that the testing is conducted on an entire system which has one or several ML-components.

ML Task

Regression
Problems with a quantitative output label [8], i.e., the predictions of the model are continuous real values. Some typical algorithms here are linear regression, ridge regression, and random forest regression.

Classification
Problems with a qualitative output label [8], i.e., the predictions of the model are discrete class. For example predicting whether an image is of a dog or a cat. Some common algorithms here are SVM, decision tree, and logistic regression (despite its name).

Structured Prediction
Models that make a structured prediction, which is not just a single value like Regression and Classification. It is instead a more complex prediction made by chaining predictions together. For example, the output could be a sentence or a tree data structure. In the text example the output could have been made by the model by chaining words together to generate the final prediction.

Clustering
The problem of identifying distinct groups within a dataset [8], i.e. dividing a larger group of individuals/items into subgroups with similarities. The output is often visualised, with each group/cluster getting a unique colour in a graph that shows how similar all items are to each other. Some typical algorithms for this is K-Means, DBSCAN, and Gaussian Mixture.

Reinforcement Learning
The problem of learning the optimal actions to take in an environment; the agent seeks to act such that the total reward received on the long-term is maximised [9]. A common algorithm for this is Q-Learning.

Dimensionality Reduction
This refers to the task of reducing the number of features (dimensions) of a dataset while removing as little information as possible [10]. Some examples of dimensionality reduction algorithms are Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA).

Anomaly Detection
For this task the goal of the ML models is to identify outliers, since they can represent things such as bank fraud or erroneous medical information [11]. Can be seen as a binary version of Classification that only aims to classify between normal and abnormal.

Ranking
The task of predicting an ordering (ranking) of objects with respect to some criterion [10]. Examples of algorithms for this is MartiRank and SVM-Light [12].

General
This tag was used when the paper in question did not state any limitations on their proposal, and there also didn't seem to be anything in the paper that would limit the use of the technique to only a specific ML Task. This tag is sometimes used in combination with the Limitation category to show that the Task is irrelevant for the proposal, but that it is limited to a certain field or data.

Access Level

Refers to pieces of information that a testing activity has access or requires access to. If the CUT is the data, then the access level is specified as not applicable (N/A). The following definitions of access levels are taken from Riccio et al. [7].

Black-box
The testing activity only has access to the input and output of the component under test. This type of testing has the most generalisability, since the underlying model does not matter. Instead the focus is purely on the inputs and outputs.

Data-box
Extending black-box access, data-box testing activities also has access to the training and test data for the component under test. This is meant to provide some insight into why the model has learned a certain behaviour.

White-box
The testing activity has access to the internal details of the component under test. For example, hyper-parameters, model weights, code, in addition to the things from black-box and data-box. This gives the developers full insight into why the model has learned a certain behaviour and how it is making its predictions.

Offline and Online Testing

Zhang et al. [2] explain the workflow of ML, where offline testing and online testing are two major phases. This section describes the difference between these. The definitions are based on Haq et al. [13], but not restricted to DNNs.

Offline Testing
Testing activities performed before the model is deployed in the system where it will be applied. This kind of testing is done on the model as a stand-alone component.

Online Testing
Online testing refers to testing activities performed on a deployed model, integrated in its application system. The system itself interacts with its environment. Generally, this implies that the component under test is the MLS.

Testing Workflow

Testing Workflow has been split into two categories in the mapping tool, since it can be important to know which part of the workflow is the focus and which parts are just being touched upon.

Test Case Design
If the testing technique has a focus on generating test cases then it will include this tag. This is by far the most common tag, both as the main workflow but also as a part being touched upon when other parts of the workflow are in focus.

Test Case Execution
If the testing technique helps the developers automatically execute tests it will have this tag. For example combinatorial testing if a tool can be used to automate execution of all the combinations.

Test Case Analysis
If the testing technique is based on analysing how efficient/effective/complete the test cases are then this tag will be included. For example mutation testing or coverage-based methods that help the developers analyse their test cases to find weaknesses and determine how to improve them.

Test Result Analysis
If the testing technique uses the results from tests to draw conclusions then this tag will be included. The decision was made to have Test Case Design take priority over Test Result Analysis as the "main workflow" if both are included in a case such as metamorphic testing.

Input Data Preparation
If the testing technique has a focus on preparing test or training data then it will have this tag. This can be data cleaning, or removing bias that can be identified without running tests.

Limitation

The tags **AD** (Autonomous Driving), **NLP** (Natural Language Processing), **Medical**, **Recommender Systems**, **Computer Vision**, **Audio Processing**, and **Graph-based data** were used when the paper in question had a proposal that was limited in what type of data it was generating or evaluating. This could for example be driving simulations for AD or audio files for Audio Processing.

Similarly, the tags **SVM** (Support Vector Machine), **K-Means**, **Neural Network**, and **Polyhedral Region** were used when the paper in question only dealt with one type of model, or in the case of Polyhedral Region [14] it is all ML algorithms that have a polyhedral region. As a clarification, it is not when the paper only shows their idea in action on one model that these tags would be used, but instead when it is shown to only work for that type of model (i.e. white-box technique).

Testing Technique

The testing techniques listed below are the techniques that were most commonly encountered during the research phase of making the mapping. However, there is also a tag called **Other Type** that is used for techniques that appeared very rarely. These techniques are then described in the *Comment* column of the mapping.

Adversarial Attack
Refers to techniques that utilise adversarial input generation, as described in [7]. This type of testing targets presumed weaknesses in models that need to be tested, most often to try to achieve the goal of Adversarial Robustness.

Boundary Value
Any paper that utilised boundary value analysis (BVA) or testing (BVT) was given the Boundary Value tag. BVA is about pinpointing different classes of input that should yield similar output, so called equivalence partitions [15]. BVT is about sampling test inputs at the boundaries between equivalence classes to verify that the system correctly handles these input values [15]. The key concern is to look at the boundaries between equivalence partitions, since these are the values that have the highest likelihood of revealing faults in the programs.

Combinatorial
Tag used for any paper that used combinatorial testing (CT), which is a special strategy for generating test cases [16]. Both configuration CT and input CT are included in this tag. The technique essentially aims to test all or a large amount of the possible combinations of input/configuration parameters to find faults or verify the correctness and completeness of the system.

Coverage-based
Refers to coverage-based (also called structural) testing, as discussed by [7]. This tag was used for papers that utilised existing coverage-based testing methods or proposed new test adequacy criteria. Some common examples used for ML is path-coverage and neuron coverage.

Differential
This tag was used when the testing technique was based on using a differential oracle as described in [7]. This can for example be synthesising a mirror program that can then be used as the oracle to determine the "correct" output.

Fuzz
Refers to a technique that aims to crash a program with a wide array of inputs [17]. Fuzz testing generally does not have to only focus on crashing the program, but this decision was made to differentiate between the Fuzz and Adversarial Attack tags.

Metamorphic Testing
Refers both to defining metamorphic relations between input and output, and to verification of these by applying metamorphic transformations on existing test input. This technique requires the developers to define metamorphic relationships, which are essentially rules for how they expect the output to change, or stay the same, when the input is modified in some specific way. The act of modifying an input whose corresponding output is known is known as a metamorphic transformation.

Mutation
Mutation testing for ML systems is often done by mutating the data instead of mutating the source code (learning program), although exceptions exist. It is also quite often combined with metamorphic testing since it can be used to evaluate the metamorphic testing by checking how many mutants it is able to kill [7].

Property-based
Property-based testing is a technique where the developer declares properties that the program, or in this case ML component, should have and then they can automatically generate a large amount of input data to test that the properties hold [18].

Search-based
Search-based testing is an application of certain search techniques, guided by a problem-specific "fitness function", to automate some testing activity [19]. It can be used generate efficient test data. Some common search techniques employed here are hill climber and genetic algorithm, the choice of which depending on how much resources the developer wants to allocate.

Regression Testing
A type of testing where the developers test the current system against a previous version to see if any behaviour has changed [7]. Regression testing has no relation to the ML task of regression. This type of testing is based on the assumption that the current system works exactly like the developers want it to, and any changes in the future that changes this behaviour would mean that a fault has been introduced.

Equivalence Class Partitioning
Refers to a technique where the input space is split into regions of inputs [15]. Input from one of these classes should then have the same or similar output when passed to the system under test, or in this case the ML component.

References

[1] Pin-Yu Chen. *Securing AI systems with adversarial robustness*. 2021. URL: https://research.ibm.com/blog/securing-ai-workflows-with-adversarial-robustness.

[2] Jie M Zhang et al. "Machine learning testing: Survey, landscapes and horizons". *IEEE Transactions on Software Engineering* (2020).

[3] Cynthia Dwork et al. "Fairness through Awareness". Association for Computing Machinery, 2012, pp. 214–226. ISBN: 9781450311151. DOI: 10.1145/2090236.2090255. URL: https://doi.org/10.1145/2090236.2090255.

[4] Matt J Kusner et al. "Counterfactual Fairness". Ed. by I Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf.

[5] International Electrotechnical Commission. International Organization for Standardization. *Systems and software engineering : systems and software quality requirements and evaluation (SQuaRE) : system and software quality models*. ISO : IEC, 2011.

[6] Arnab Sharma and Heike Wehrheim. "Higher income, larger loan? Monotonicity testing of machine learning models". *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2020, pp. 200–210.

[7] Vincenzo Riccio et al. "Testing machine learning based systems: a systematic mapping". *Empirical Software Engineering* 25.6 (2020), pp. 5193–5254.

[8] Gareth James et al. *An introduction to statistical learning with applications in R*. Springer, 2021.

[9] Richard S. Sutton, Francis Bach, and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.

[10] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[12] Christian Murphy, Gail E Kaiser, and Marta Arias. "An approach to software testing of machine learning applications" (2007).

[13] Fitash Ul Haq et al. "Comparing Offline and Online Testing of Deep Neural Networks: An Autonomous Car Case Study". *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. 2020, pp. 85–95. DOI: 10.1109/ICST46399.2020.00019.

[14] Md Sohel Ahmed, Fuyuki Ishikawa, and Mahito Sugiyama. "Testing machine learning code using polyhedral region". Association for Computing Machinery, Inc, Nov. 2020, pp. 1533–1536. ISBN: 9781450370431. DOI: 10.1145/3368089.3417043.

[15] Felix Dobslaw, Francisco Gomes de Oliveira Neto, and Robert Feldt. "Boundary value exploration for software analysis". *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2020, pp. 346–353.

[16] D Richard Kuhn, Raghu N Kacker, and Yu Lei. *Introduction to combinatorial testing*. CRC press, 2013.

[17] Barton P. Miller, Louis Fredriksen, and Bryan So. "An Empirical Study of the Reliability of UNIX Utilities". *Commun. ACM* 33.12 (1990), 32–44. ISSN: 0001-0782. DOI: 10.1145/96267.96279. URL: https://doi.org/10.1145/96267.96279.

[18] George Fink and Matt Bishop. "Property-based testing: a new approach to testing for assurance". *ACM SIGSOFT Software Engineering Notes* 22.4 (1997), pp. 74–80.

[19] Phil McMinn. "Search-based software testing: Past, present and future". *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. IEEE. 2011, pp. 153–163.