Paper summaries

Alexander Simola Bergsten,
Erik Bock

April 7, 2022

This paper contains summaries of all the papers in the mapping. The purpose is to provide insight into the papers similarly to an abstract, so that the user of the mapping can get an overview of the proposed solution in the paper before reading it completely. Every paper gets one section, and the title of the paper is written in bold at the beginning of the section. The sections are ordered alphabetically with regards to the paper title.

A systematic approach for evaluating artificial intelligence models in industrial settings: A systematic approach to test the robustness of a trained model is presented in [1]. It injects perturbations in a test data set, aiming to expose model vulnerabilities in the form of input that the model handles insufficiently well. Two kinds of perturbations are defined in the paper: "swapping", i.e., altering the sequence of consecutive data points; and "dropping", i.e., removing data points. These are applicable to time-series data and mimics the data degradation that can occur in a realistic setting. The authors assess the method on 20 datasets for 7 state-of-the-art algorithms. They conclude that the perturbations are effective at challenging the algorithms, to different degrees, based on the results that the model accuracy is lower on the perturbed than on the clean test data. Furthermore, the specific perturbations that should be used to evaluate model robustness in any circumstance depends on the industrial context and domain. The authors also investigate if the robustness can be predicted before applying the systematic approach described above. They trained a set of decision trees on metadata about each dataset, the parameters of the perturbation used, and a label. The ground truth label corresponds to how much the algorithm was affected by the perturbations, i.e., a measure of the robustness. It was found that this was not a reliable method to predict robustness, so the authors recommend the original approach of generating perturbations instead.

Application of metamorphic testing to supervised classifiers: Xie et al. [2] presents more MRs for classifiers, while also showing real world results from a case study the authors have conducted. Some examples of MRs that are recommended in the paper are permutations of class labels or the removal of a class completely. In the first case the we want to assure ourselves that if we have a function F that maps each type of label to another type of label, then if we have data D and output O in the original run then the next run, given that the mapping function is applied to all labels, should output F(O) given the same data D. In the second case, where we remove a label, we first have to make sure that this is not the label that is predicted by the model in the first run. Once this is confirmed we can remove any other label before the second run, and the prediction should remain the same since the removed label was not the one predicted in the first run.

Assessing the robustness of conversational agents using paraphrases: The act of using paraphrases to test the robustness of conversational agents, such as Amazon Alexa or Google Home, is proposed by Guichard et al. [3]. This concept is similar to using Adversarial Attack testing where the ML classifier is given modified data that is meant to make it classify it incorrectly. In the case of the paper they aim to take a sentence that has been said and change some of the words used into synonyms. This enables generation of a large number of new sentences that can be used to test the robustness of the conversational agents. The authors perform a case study and the results indicate that using paraphrases when training can lead to a more robust model.

Astraea: Grammar-based fairness testing: Soremekun et al. [4] proposes another testing approach called ASTAREA that is meant to help developers identify issues with fairness in their NLP models. ASTRAEA uses context-free grammars (CFG) to generate input that include biases that can reveal these issues. The authors state that the idea is that in the generated input there is sentence a and b and the only difference between these is a protected attribute, such as gender or occupation. If the output of a model differs for the input of sentence a or b, then that would constitute a fairness violation where a protected attribute affects the models prediction. ASTRAEA is evaluated against 18 models, and the results show that it on average improves the fairness of those models by 76% according to the authors. One of the limitations of ASTRAEA is that it is only able to detect fairness violations that are in the grammar. This means that the developer needs to have an idea of what fairness violations might occur.

Automated directed fairness testing: Udeshi et al. [5] implemented the automated tool Aequitas that can generate test inputs to check the predictive fairness of a model, and consecutively retrain the model, with the gathered inputs, to improve its fairness. To find an initial set of discriminatory inputs, Aequitas first performs a uniform random search over the input space. It then searches locally around those inputs to extend this set. The hypothesis is that if there exists a discriminatory input I, then more such inputs exist in the vicinity of I. The authors recognize that this assumes that the model under test is consistent (referring to noise robustness). Compared to the baseline of only selecting random inputs, the authors show that Aequitas performs approximately 20 times better, in terms of the number of discriminatory inputs it finds on the same duration of time. The paper provides algorithms for global and local search on the input space, and for retraining the model.

Automatic fairness testing of machine learning models: Sharma and Wehrheim [6] proposes Verification-based testing (VBT) for testing the fairness of classifiers. The authors state that the technique is based on both metamorphic testing and property-based testing, since it can only be tested on pairs of input like metamorphic testing and that the user can specify fairness requirements/properties for which test cases and data then will be automatically generated for like in property-based testing. The idea with VBT according to the authors is to approximate the behaviour of the original black-box model with a white-box model. The white-box model in this case would be a decision tree, since the authors state that it can easily be converted into logic formulae which makes it easy to verify properties. The white-box is trained on input/output combinations from the original model, and then it is tested for fairness with the help of an SMT solver which allows them to generate more data. This new data can be tested on the original model and help the developers identify fairness violations. It is then possible to re-train the white-box model with these new data and start the process again.

Automatic system testing of programs without test oracles: Murphy et al. [7] proposes Automated Metamorphic System Testing to help remedy the problems of manual metamorphic testing like the results being hard to interpret by a human or that there might be several correct answers. To make this a reality the present the framework Amsterdam which allows a developer to configure the metamorphic tests and what to do if one of them fails. The authors also introduce Heuristic Metamorphic testing to deal with the issue of metamorphic testing resulting in a lot of false positives. The most common issue according to the authors is when there are floating point calculations, so the idea with heuristic metamorphic testing is that a test should pass as long as it is "close enough" to the right value, even if it's not exactly the same.

Automatic testing and improvement of machine translation: Sun et al. [8] presents and automatic approach to testing called TransRepair. This approach utilises both mutation testing and metamorphic testing to identify and repair bugs in Machine Translation systems. The approach is black-box, with a grey-box alternative, and the authors liken it to a post-processing phase of the testing phase where it is possible to target and fix specific bugs. The three automatic main steps in the approach are test input generation, test oracle generation, and inconsistency repair. Test input generation is done through mutating sentences by replacing words with synonyms, and then filtering out any sentences that are no longer grammatically correct. Test oracle generation is done through metamorphic relationships. Inconsistency repair is done when a inconsistency is identified, i.e. a mutant does not result in the same translation as other mutants from the same base sentence. The authors deem that TransRepair is effective at locating and repairing bugs.

Automatically Authoring Regression Tests for Machine-Learning Based Systems: A methodology of automatically authoring regression tests has been proposed by Zhu et al. [9]. Their proposal is implemented for a spelling checker system, but the authors claim that it is generalisable to many kinds of ML-based systems. The motivation for the tool stems from some major challenges in the field of regression testing for ML, all of which are outlined in the paper. One challenge is the degradation of test cases in the test suite as the oracle gradually turns obsolete. This is handled by periodically updating the test suite with data from the system in use, where the oracle is inferred from user feedback and interaction with the SUT together with perturbations. To obtain some form of coverage on the test input space, the authors learn a coverage-driven perturbation model. In the end, the paper is quite specific to its application domain of spell-checking, but serves as guidance for creating such an automated test suite for other ML-based systems.

Automatically detecting data drift in machine learning classifiers: Ackerman et al. [10] discuss the phenomenon they call *data drift*, which refers to the gradual change of the production data distribution leading to decreased model performance. The authors propose an approach for determining the amount of data drift at a given time by looking at the predictions of a classifier and its confidence in those. A new concept called *auditors* is introduced in the paper, referring to the mechanism used to detect drift, and a concrete auditor is implemented specifically for classifiers. To define a custom auditor, the authors describe a framework for both defining it and evaluating its ability to detect drift via a simulation environment. This approach is useful when labeled data is lacking, expensive to get, or when deep understanding of the domain is needed to do correct data labeling.

Automating large-scale data quality verification: Schelter et al. [11] seek to automate the process of validating large sets of data in a continuous way. They contribute a declarative API that enables the creation of unit tests for data. Users of the API can specify quality constraints on their data, which are assessed by computing the relevant metric for the given quality attribute. The user need also write the code that validates the constraint, however, they do not need to implement the code that computes the metric as this is automated by the authors. A set of quality attributes, including consistency (the degree to which a set of semantic rules are violated), completeness, and accuracy, are considered in the paper. The authors implement an incremental computation of different metrics related to those quality attributes, where a relatively small dataset is required to update the quality metrics because the system maintains state of the previous computations. This improves the scalability towards large datasets in comparison to batch computation which iterates over all observed data to compute the metrics. An approach like this could be used in online testing of the data for a given ML pipeline. Concluding, different ways of doing anomaly detection are

possible with this framework. The authors define some simple pre-defined algorithms but also lets the user write their own algorithms to accomodate different domains and applications.

Beyond Accuracy: Behavioral testing of NLP models with CheckList: In [12] the authors discuss a tool called CheckList that can help developers test that their models are able to fulfil more quality criteria than just making accurate predictions. This can help reveal faults with the models that affect for example their fairness, when it comes to making worse predictions for certain subsets of people, or their robustness, when it comes to not being able to properly parse certain input like negation.

Black box fairness testing of machine learning models: Aggarwal et al. [13] propose an automated test input generation method for assessing the degree of individual discrimination, as opposed to group discrimination, exhibited by a model. It is based on the techniques symbolic execution and local explainability. The authors compare their algorithm SG to Themis [14] and Aequitas [5], and run experiments against these in terms of the number of discriminatory test cases they generate. The results indicate that SG outperforms the other two, although the experiment sample size is just one.

Boostclean: Automated error detection and repair for machine learning: Krishnan et al. [15] proposes a new tool called Boostclean which is meant to help developers find and repair faults in the input data that models would be trained on. The idea behind the tool is that it can identify a wide array of domain value violations, which are for example missing values. If a model is unable to handle these violations, the it would lead to worse accuracy and less consistency. The authors idea is that Boostclean will help developers automatically identify and repair common faults, so that the developers themselves can focus on ones that might be specific to their own domain. Boostclean also comes with some possibilities for customisation to further help the developers, like several different simple repair functions, which for example replaces the faulty value with the mean or replaces it with the median value of the dataset. By doing this they believe it will be easier to create robust models.

Bridging the gap between ML solutions and their business requirements using feature interactions: In the paper by Barash et al. [16], a method is presented for connecting ML solutions to an enterprise's business requirements in a more structured way through combinatorial testing. This can reveal weak areas where a model does not live up to a quality requirement, which can then be addressed. The authors state that this is meant to bring new opportunities for testing ML, which are sometimes overshadowed by just testing the predictive accuracy. The method is meant to be applied in the industrial setting and the paper provides three case studies that details the results of the application. The results indicate the the method is a good way of gaining insight into the quality of ML solutions according to the authors.

Can We Achieve Fairness Using Semi-Supervised Learning?: Another way of improving fairness was presented by Chakraborty et al. [17]. In this paper the authors propose a framework for helping improve fairness through semi-supervised learning techniques. Semi-supervised learning differs from supervised learning in the sense that only parts of the data is labelled, in the papers case around 10%, and the remaining data has to be given pseudo-labels generated from the original labels. The framework presented in the paper helps make this process of generating labels as balanced as possible, so that when a classification model is then trained with the data it will be more fair. The authors also state that the framework is model agnostic.

Chatbot Testing Using AI Planning: Bozic et al. [18] proposes a framework for testing the communication capability of chatbots. It utilises the concept of AI planning, where a specification is written in Planning Domain Definition Language. The specification is split into a domain file and a problem file. It can then be given to a planner that generates plans which can then be executed as tests on the chatbots. Currently the authors state that the framework only supports HTTPS chatbots, but that a new interface could be created to support other types. When a plan is executed the framework waits for a response from the chatbot, in the example case a reservation booking, and if it gets one it will be tested against a test oracle. After the specification is written this entire process is automatic according to the authors, and can help automate functional testing for chatbots. The framework shows promise according to the authors, but there are still problems with misunderstandings that need to be addressed.

Combination and mutation strategies to support test data generation in the context of autonomous vehicles: For testing autonomous driving systems, Neves et al. [19] developed a genetic algorithm that can be used to generate test data. The authors propose ways to generate new offspring for such an algorithm. Specifically, they contribute with combination and mutation strategies. Their test generation strategy aims to reach a certain degree of coverage on the system. While this paper is focused on the domain of autonomous vehicles, the approach of genetic algorithms to generate test cases might be generalisable to other applications.

Combinatorial Testing Metrics for Machine Learning: Lanus et al. [20] propose a set difference metrics that is meant to be used as an indication of how similar two ML datasets are. The authors utilise concept of combinatorial testing to create this metric, and argue that by knowing how similar two datasets are we can predict how well a model that is trained on one of them will be able to perform on the other. The authors state that the metric can first and foremost help with fault localisation, since combinatorial testing can be used to narrow down which combinations of attributes are present in failing but not passing test cases. Another application of the metric would be to help with the interpretability and explainability of classifiers. This is similar to the case with fault localisation, where the authors are looking for a subset of factors that can help uniquely identify a class. Other than those two cases it is also possible to identify combinations of factors the occur often in the input, which would imply that the model has been trained well for these but also that there might be an issue of overfitting. It is then possible to look at it from the other direction as well, and try to find more input data for rarer combinations of factors for which the model has been trained less.

CrossASR: Efficient Differential Testing of Automatic Speech Recognition via Text-To-Speech: Asyrofi et al. [21] present an approach called CrossASR to dealing with the issues, like time and resources limitations, of manually creating tests for automatic speech recognition (ASR) systems. The core concept is to use differential testing by testing several ASR systems against each other. A goal of CrossASR is also to generate test cases as efficiently as possible, by generating the least number of cases necessary to reveal failures. To be able to do this CrossASR has failure probability predictor in the form of a classifier that is able to predict which text has the highest probability of revealing a fault in an ASR. Each time the classifier generates test cases it is also trained, so that it in subsequent iterations keeps being accurate. The authors state that CrossASR keeps iterating and generating test cases until one of four things happen. The first is that it has generated enough failing test cases to each the goal, the second is that it runs out of a time budget, the third is that it reaches the max number of iterations, and the final is that it has processed the entire text collection.

The authors state that CrossASR functions for all ASR since it is black-box testing, and the results show that it can effectively test the reliability of ASRs.

Data Synthesis for Testing Black-Box Machine Learning Models: In the paper by Saha et al. [22] they present a framework for synthesising input data that can help test both traditional models and deep learning models. The goal is to increase trust for the machine learning models through increasing the robustness and fairness of them. To generate realistic data they have to set constraints that must be fulfilled, like if the data should be taken from a discrete or continuous distribution, and if it's numeric or categorical. It's also possible for the user of the framework to set their own constraints that can override the initial ones.

Data Validation for Machine Learning: An industrial case of a data validation system is explained by Breck et al. [23]. It was designed to tackle and prevent the problems posed by feeding bad data into ML pipelines. The authors took inspiration from software unit testing and database schemas. The system provides three kinds of data validation: single-batch validation, inter-batch validation, and model testing. The ML pipeline in the paper receives batches of data regularly, which triggers the data validation system on every reception. For single-batch validation the data validation system analyses the batch for data errors. The data is validated against a schema, which is updated regularly to cope with the changing characteristics of the data. Inter-batch validation aims to capture data anomalies in the form of unexpected changes between two or more batches. The specific metric used to detect distribution changes was chosen pragmatically so as to simplify interpretation and tuning of thresholds. Finally, model testing is done to check for deviations between the expected data and the assumptions of the training code. It is about finding and dealing with hidden assumptions used in the learning program. The authors employ a kind of fuzz testing to generate test inputs that follow the schema constraints. As for the applicability of this data validation system, the paper does not state any strict limitations such as to only deep neural networks or only supervised learning.

Datamorphic testing: A method for testing intelligent applications: "Datamorphic testing: A method for testing intelligent applications" [24] demonstrate a new method for generating test cases, called datamorphic testing. They apply the method on facial recognition software. Mathematical definitions on the terms datamorphism, metamorphic relation, and metamorphism are provided in the paper. Several strategies of using datamorphic testing to generate new test cases are outlined as well, such as explorative or combinatorial generation.

Dataset Coverage for Testing Machine Learning Computer Programs: Nakajima and Bui [25] present a way to derive Metamorphic Relations (MRs), which are meant to replace other more ad hoc methods that are currently being used. The authors target classifiers, and while the paper does discuss specifically Support Vector Machines (SVM) this is only to provide an example case of the usage of the method. The goal is to generate a pseudo-oracle through the MRs, and to allow the learning programs that creates models to be tested.

Fairness-Aware Programming: In [26] the authors introduce a new concept for programming decision-based systems that need to take fairness into account. They propose letting the programmers themselves state what the fairness expectations of the systems are, and provides general specification language for making this possible. The language has a context free grammar (CFG) representation, but the authors also provide a concrete Python implementation following the decorator design pattern. The idea is to let the programmer define specifications, fairness definitions, for each procedure that it must then follow. During runtime, the system's decisions are validated against

these programmed fairness checks and reports are generated when fairness if not adhered to.

FairTest: Discovering Unwarranted Associations in Data-Driven Applications: Fairtest is a tool presented by Tramer et al. [27] which is meant to help developers identify fairness issues in their systems. The tool is the realisation of a framework called Unwarranted Association, which was also created by the authors. Unwarranted associations is also the name the authors have given to all types of fairness bugs, and the goal of the framework is to find and analyse these bugs. The tool seems to be model-agnostic, since it's a black-box approach that only looks at the input and output of a data-driven application (for example a ML system). The main utilities that the tool provides according to the authors are error profiling and discovery/testing of association bugs.

Generalized Oracle for Testing Machine Learning Computer Programs: Nakajima [28] proposes using metamorphic testing as a generalised oracle for ML systems. The paper brings up examples of this metamorphic relations (MR) for Support Vector Machines (SVM) and Neural Networks (NN). For SVM the MRs are based on duality optimisation of a Lagrangian, which for example means that switching the indices of two data points would not affect the end result, which can then be turned into a MR for testing. For NNs on the other hand the same MR would not work as for SVM due to the underlying representation of the models. For an NN a behavioural oracle could instead come in the form of the multiplicative property, where each data point is multiplied with the same constant which results in a change of the shape of the indicator graphs.

Generating Adversarial Driving Scenarios in High-Fidelity Simulators: Abeysirigoonawardena et al. [29] present a way to generate AD scenarios for simulations, with the added benefit of making the simulated pedestrian behave in an adversarial fashion. This means that it can reveal faults that would have gone unnoticed otherwise. Without going into too much detail, the essence of the generated adversarial behaviour is that over time pedestrians will end up moving towards the car's location. Bayesian optimisation is used to find the most optimal adversarial scenarios. The authors' goal is for this type of testing to be able to improve the reliability of cars on the road.

Higher income, larger loan? monotonicity testing of machine learning models: Monotonicity is a ML systems ability to understand connections between attributes as defined by Sharma and Wehrheim [30]. They use the example of the attributes income and chance to get a loan, since according to the authors the model should be able to create the correlation that a higher income would increase the chances to get a loan. If the input to the model is the income and it is increased, then the output, which would be the chance to get a loan, would also increase. If this is not the case, then the model would have low monotonicty.

Identifying and Mitigating Spurious Correlations for Improving Robustness in NLP Models: Wang et al. [31] claims that many issues with robustness stems from the fact that models take "shortcuts" in their predictions, meaning that they learn spurious correlations that are not really there. In their paper they therefore aim to address these issues with the help of a framework that can identify spurious correlations so that they can be mitigated. The framework uses interprability methods that allows it to extract tokens that an NLP model found important for its decision making. The framework then splits these tokens into a "genuine" group and a "spurious" group by checking how stable a model is if these tokens are replaced with semantically similar words. The final step is then to use the spurious tokens to target the shortcuts the model is taking and preventing it from using them. Based on their experiments the authors say that the framework is able to improve an NLP models robustness for out-of-sample data.

Identifying implementation bugs in machine learning based image classifiers using Metamorphic Testing: Dwarakanath et al. [32] proposes more metamorphic relationships to help test both SVM and CNN based applications. For the SVM-based application, which is meant to identify a hand-written digit, the MRs are for example permutation of training and test features or the order of training instances. For the CNN-based application, which is an image classifier, the MRs are for example permutations for the input RGB channels for the training and test data as well as permutations for the convolution operation order. The authors introduced bugs into both of the applications, creating mutants, and tested how well the MRs were able to identify them. The MRs were able to detect all the mutants for the digit classifier, and half the mutants for the image classifier.

Improving question answering model robustness with synthetic adversarial data generation: Another approach to dealing with the robustness was presented in a paper by Bartolo et al. [33]. The paper presents a way to synthetically generate adversarial input data to be able to test the robustness of question answering models. This is done through the use of another model that can take a smaller adversarial dataset and generate a much larger dataset of relevant questions. In the paper it is only demonstrated for DNNs, but the solution seems general enough to be applied to traditional ML as well since it does not depend on any implementation details of the models. The solution itself utilises several both $BART_{Large}$ and RoBERTa models to generate the adversarial data.

Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision: In the paper by Ramanathan et al. [34] they present a new way to perform symbolic analysis and statistical testing. The symbolic analysis helps the authors identify test cases for which the model is correct, but for which even a small change to the input can lead to failures. They then also use statistical hypothesis testing to be able to determine if a test case is able to tell a correct and faulty system apart. They also show how specifically a K-Means model and a Computer Vision model can be tested, with distance-theoretic abstractions and the OpenCV human detection algorithm respectively.

LAMP: Data Provenance for Graph Based Machine Learning Algorithms through Derivative Computation: LAMP is a provenance computation system created by Ma et al. [35]. It is based on the concept of automatic differentiation, and allows for the calculation of a derivative between the input and the output of a model. A high derivative would mean that the input has a high impact on the output, and this is useful information since it means that small perturbations to this input will heavily affect the output. The resulting system LAMP proves to be a useful tool for optimising ML models and help debugging both the models and the data.

Measuring and improving consistency in pretrained language models: Elazar et al. [36] discussed the importance of consistency for Pretrained Languages Models (PLM), where the behaviour of a model should not change unless the meaning of the input has been changed. This means that it should be able to handle changes to a sentence that do not affect the meaning of the sentence. The authors have created a resource called ParaRel that helps them identify consistency issues in many existing PLMs, focusing on the BERT family of models. Since the ParaRel evaluation revealed many consistency issues, the authors recommended adding another step to the pretraining phase of model training where the consistency attribute should be manifested. This involves training the models on both relationships between words and paraphrasing, which in the end helps improve the consistency of the models.

Metamorphic Testing for Plant Identification Mobile Applications Based on Test Contexts: Guo et al. [37] propose an automated testing approach called *TestPlantID* to test mobile apps that have a plant identification feature. TestPlantID uses so called test-context-based MRs, the meaning of which is elaborated by an example where the same plant (test subject) should not be identified differently depending on the angle of photography, for example. Another MR that should hold is that the background (context) in the image depicting a plant should not impact the prediction. The authors define seven different test contexts, i.e., certain variables that should not impact the output. The first example above represents the Background test context, while the second corresponds to Angle.

Metamorphic Testing for Quality Assurance of Protein Function Prediction Tools: Metamorphic testing has also been applied to test automated protein function prediction tools [38]. The authors identify metamorphic relations used to generate new test cases for such tools based on existing test inputs.

METTLE: A METamorphic Testing Approach to Assessing and Validating Unsupervised Machine Learning Systems: METTLE: A METamorphic Testing Approach to Assessing and Validating Unsupervised Machine Learning Systems [39] is a paper that focuses on a general solution for testing unsupervised ML systems, more specifically clustering, using metamorphic testing. The goal is the correctness, robustness, and completeness of the system, and the way the authors propose achieving it is by using their methodology METTLE that allows for creating metamorphic relations (MR). METTLE comes with 11 predefined MRs, but the users are recommended to only use the ones they feel are a good fit for their use case. They also state that the users can create new MRs if needed. One example of an MR included in METTLE is that given a sample, the clustering result should remain the same even if the order of objects in the sample change. The authors also state that other than just evaluating a clustering system, it's also possible to use METTLE to help the user select the best clustering system for their purposes.

Multiple-implementation testing of supervised learning software: Srisakaokul et al. [40] proposes an interesting approach to dealing with the oracle-problem for classifiers. The authors describe how to utilise several different implementations for determining what the "correct" result would be, and then uses a majority-vote to determine which is the most likely to actually be the correct result. The reason they utilise several techniques is that all of the techniques have their own weaknesses and limitations, so the authors argue that the techniques together can help mitigate those problems. In their evaluations they identified more faults with their multiple implementation testing than the benchmark oracle they were comparing it with, and had a much lower false positive rate. The paper does not put much spotlight on the multiple implementations that are used for the testing, since the paper focuses more on proving the concept of multiple implementation testing rather than finding the best combination of implementations.

On Using Decision Tree Coverage Criteria for Testing Machine Learning Models: Santos et al. [41] propose two white-box test adequacy criteria to use when generating test cases for ML-based systems. The first criterion is called decision tree coverage (DTC), which says that all possible root-to-leaf paths should be traversed in a decision tree trained on the same data as the model under test. The second criterion is referred to as Boundary Value Analysis (BVA) and it states that the test cases should cover valid boundary values of decision nodes. The overall goal of these criteria is to facilitate the task of assessing the effectiveness of tests for ML-based systems, and

consequently to help testers generate good tests that could highlight weaknesses of a model, so that they can be addressed. The authors evaluate the effectiveness of the criteria themselves by applying them to testing a K-Nearest Neighbours model, the results indicating a significant difference in effectiveness compared to the baseline of randomly sampling the input space for test cases.

Perturbation Sensitivity Analysis to Detect Unintended Model Biases: A framework for detecting biases of an NLP model's predictions is the Perturbation Sensitivity Analysis (PSA) by Prabhakaran et al. [42]. Two tasks within NLP are studied by the authors, sentiment and toxicity analysis. The framework is generalisable across all tasks in the NLP domain, but does not apply to autonomous driving or medicine, for example. PSA relies on the notion of perturbations, in this case input sentences are altered by swapping a word corresponding to a named entity for another, which should not meaningfully impact the predictions of the model. So this is an application of metamorphic testing to assessing fairness.

Properties of Machine Learning Applications for Use in Metamorphic Testing: One of the earliest papers about using metamorphic testing to deal with the oracle problem of ML testing is the work by Murphy et al. [43]. They analyse three ML algorithms (SVM-Light [44], PAYL [45], and MartiRank [46]) to find metamorphic properties potentially shared across many algorithms. Six common properties were found, for example the invertive property which means that it is possible to predict the output of a given input's opposite value. The remaining five were called the additive, multiplicative, permutative, inclusive, and exlusive.

Property-Based Testing for Parameter Learning of Probabilistic Graphical Models: Saranti et al. [47] proposes using Property-based testing (PBT) to test ML algorithms, and shows a concrete case of PBT on a parameter learning algorithm. The reason PBT should be used according to the authors is that it makes it possible to generate large amounts of test data by setting which properties that data should hold. This then allows the the authors to test the parameter learning algorithm expectation-maximisation (EM), since the number of test cases that have to be executed is large. Important to note here is that EM should not be ran for too many iterations, since that will make the estimations of the parameters become overfitted to the test dataset. The authors conclusion is that PBT is a good fit for white-box testing of ML algorithms.

Robustness gym: Unifying the nlp evaluation landscape: Robustness Gym is an evaluation toolkit aimed at NLPs proposed by Goel et al. [48]. While the paper itself only mentions deep learning ML, there is no indication that the toolkit would only work for DNNs and not also traditional ML models. The goal of the tool is to increase the robustness of models, and they try to achieve this with four different actions. Firstly they can identify subpopulations for which a model performs poorly. Secondly they apply transformations to the data to see if the model can keep up with the changes. Thirdly they utilise adversarial attack testing to see if the model has any weaknesses that can be exploited. Finally they also have evaluation sets that the user can utilise for either generic or targeted evaluation of the model. Robustness Gym comes with default setting that will let novice user easily use the tool according to the authors, but there is also a lot of room for customisation for more advanced users.

Robustness testing of language understanding in task-oriented dialog: A model-agnostic toolkit called LAUG is proposed by Liu et al. [49] that can help developers test the robustness of their systems dealing with natural language. LAUG aims to as accurately as possible mimic real world perturbations, which help with the otherwise time consuming task of collecting real data which

in some cases is not possible to scale. In the paper they showcase the toolkit being applied to both classification and generation LU models, and also state that the tool can be further extended in the future.

Smoke testing for machine learning: simple tests to discover severe bugs: Herbold and Haar [50] investigated how existing techniques from software testing could be applied to ML testing. They implemented so called smoke tests for both supervised (classification) and unsupervised learning (clustering) algorithms. Smoke tests check that the most essential functionality of a system is working while ignoring the details. For example, one test case defined in [50] was checking if classification/clustering implementations can correctly handle common probability distributions. There is no oracle employed in this approach. The authors consider a test case to fail if it causes the system or component under test to crash, and otherwise it passes. An approach that generates a number of test cases linear to the number of hyperparameters of an algorithm is outlined in the paper. The test input generation is inspired by combinatorial testing, boundary value analysis and equivalence class analysis, which are common techniques from software testing at large. The authors evaluated their testing approach on three frameworks (Scikit-learn, Apache Spark, and Weka), and found eleven bugs across all.

Stability evaluation for text localization systems via metamorphic testing: Yan et al. [51] proposes six new MRs that are meant to help test the stability of text localisers, while the paper also introduces a metric to be able to measure this stability. For the first two MRs they introduce the model should give the same output before and after the MR has been applied. The corresponding metamorphic transformations are: to increase the brightness of the image, and to switch the input RGB channels. There are then three more MRs where the authors expect the output to change in a certain manner. For the MR Perspective Transformation where the input is distorted, the authors want to have the same number of text segments in the output, but their locations do not have to be the same. For the MR Watermarking they add another text label to the image, so for this MR they want the output to have more text segments identified than the original. The opposite is then the Masking MR which covers an existing text label, which should result in the output having identified less text segments. All these relations were an attempt to replicate real world scenarios. The authors state that the results show that their MRs are able to evaluate the stability of the text localisation models.

Statistically rigorous testing of clustering implementations: Yin et al. [52] propose an approach for testing both stochastic and deterministic clustering implementations. The authors apply the tool on algorithms like K-Means, DBSCAN, and Gaussian mixture. It bases much of its analysis on different statistical hypothesis tests. For example in assessing the variation of a clustering algorithm's output across runs and across toolkits, as well as determine if accuracy varies across algorithms for a given toolkit. Furthermore, it can also compare different toolkits in their disagreement on which label is correct for a given algorithm. They measure accuracy of a clustering algorithm on test data with the adjusted Rand index (ARI).

Storm: Program reduction for testing and debugging probabilistic programming systems: Dutta et al. [53] presents a framework called Storm that can aid developers in optimising their programs and also with debugging them. When a failure occurs in a probabilistic program, Storm is able to create a smaller version of the program, data, and arguments that can trigger the same failure as the original program did. The authors state that Storm is language agnostic, since they have created their own intermediate representation, Storm-IR, which all analysis and transformations are

performed on and that can then be turned back into the original language. In the paper they show that they currently support both the Pyro and Stan languages.

SynEva: Evaluating ML programs by mirror program synthesis: Qin et al. [54] presents an approach called SynEva that is meant to help developers deal with the oracle problem for ML classifiers. It helps ML developers verify if their models perform as well in deployment as in training. The idea is for SynEva to synthesise a mirror-program that can then be used as a pseudo-oracle. If the mirror-program behaves differently than the trained model, there is a reason for concern. The process is entirely automated, and SynEva creates the mirror-program by mimicking the behaviour of the original program based on the knowledge gained from training scenarios (a model). SynEva's workflow can be broken down to three phases. The first is preparation, where they run a learning program to extract knowledge from training scenarios so that it can be used within what they call a knowledge program. Then comes synthesising, i.e., creation of the mirror-program, which is a joint effort of the knowledge program and both the knowledge program and training dataset. Measuring is the final phase, where the mirror-program and the knowledge-program are compared to check which scenarios their behaviours differ, e.g., in terms of predictions. The results from this final phase is put into a report that can then be used to validate the program.

Telemade: A Testing Framework for Learning-Based Malware Detection Systems: In the domain of malware detection, the kind of metamorphic testing that is done for image classifiers for example, cannot be analogously employed, according to Yang and Xie [55]. This is because by perturbing the features of a valid test input you might lose the constraints that are inherent to malware code. The input to a malware detector consists of malware programs, i.e., executable software, and this kind of data differs much from images. The authors present, for malware detectors, how to generate test inputs, how to validate such inputs, and also discuss some testing metrics to use. Their generation strategy is twofold. First, they employ an "evolution strategy" where they imitate the evolution of real malware, based on domain experience. Second, they use a "confusion strategy" where they try to imitate such malware that is uncommonly recognised by malware detectors. To ensure that the mutated input is valid, the authors once again present two methods; either they analyse the impact of the mutation, or they analyse the robustness of the malware post-mutation using random testing. The authors point out that the testing criterion one can use with Telemade could be neuron coverage [56], or the authors' own criterion called neuron-combination coverage, but also many others depending on the ML model.

Testing acoustic scene classifiers using Metamorphic Relations: Moreira et al. [57] has applied metamorphic testing to acoustic scene classifiers. They implement a wide variety of traditional ML algorithms, like SVM, k-NN, Naive Bayes, Decision Trees, and Random Forests to be able to show how effective their MRs are on these classifiers. The authors propose three MRs that are all related to sound waves, and their names are Amplitude, Noise insertion, and Shift. For Amplitude they have a function that can change the amplitude of a audio signal with a fixed value. Noise insertion is just like the name implies insertion of noise into a signal, but the classifier should still classify it as the same class as without the noise. Shift simply shifts the audio signal a fixed time, but that should not affect the classification either. The authors state that since they took a black-box approach to the testing, their MRs should be applicable to any system given that the input data is audio signals.

Testing and validating machine learning classifiers by metamorphic testing: Another early paper that proposes metamorphic testing to cope with the Oracle problem of ML is the work by Xie

et al. [58]. They apply the technique on classification algorithms, for example K-Nearest Neighbours and Naive Bayes, and provide a list of candidate metamorphic relations that can be checked. Besides showing the effectiveness of metamorphic testing, the authors argue that verification by cross-validation is not enough to assure the quality of classifiers. The technique is partly evaluated by checking the capability of the test cases to detect mutants (referring to mutation testing), which proved to be high.

Testing Framework for Black-box AI Models: A black-box framework has been developed by Aggarwal et al. [59]. It is called *AITest* and can generate test cases for assessing accuracy, robustness, and fairness of a trained model. Different data types are also supported by the framework, including tabular, textual, and time-series data.

Testing machine learning algorithms for balanced data usage: Sharma and Wehrheim [60] research how fairness testing can be done when the model is being trained, as opposed to after training. Their contribution is a framework called TILE for checking if an algorithm is balanced. They define an algorithm as being balanced, w.r.t. a specific metamorphic relation, if it produces equivalent predictors both with and without transforming the training data. Four different metamorphic transformations on tabular data are defined, such as changing the order of the columns or renaming the classes of a feature.

Testing machine learning code using polyhedral region: A generic approach of testing the implementation of an ML learning program is proposed by Ahmed et al. [61]. It is similar to equivalence class testing in that it identifies a partition, called the polyhedral region, of the input space that should produce the same output. The authors evaluate the technique on the lasso regression algorithm, stating that it manages to kill 98% of mutants introduced. They claim that testing by polyhedral region is not specific to the lasso algorithm, but many others where the model is sparse (with few features).

The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets: The concept of data linters are described and implemented by Hynes et al. [62]. It is based on the notion of code linters, which are a kind of automated code analysis tool that reports on bad aspects of source code, such as violations against coding conventions. The authors define a data linter as "a tool that analyses training data features, with respect to a specific model type, and provides recommendations for how individual features can be transformed to increase the likelihood that the model can learn from them" [62, pp. 2-3]. Their implementation in the paper is specific to DNNs, however, the idea is generalisable to other models. The evaluation of the tool suggests that it can help developers in achieving better predictive performance, especially to novice ML developers.

The importance of being external. methodological insights for the external validation of machine learning models in medicine: Cabitza et al. [63] states that one of the most important aspects of ML in the Medical field is the external validation. The reason for this is that it is common for models in the medical field to get overfitted to their data and perform worse out-of-sample. Since there already exists many methods for external validation the authors instead perform what they refer to as meta validation, which is trying to determine how well external validation actually works. To do this they have designed two graphical tools that help them assess the robustness of a model based on the data similarity as well as how well the external validation performed. They perform an experiment with COVID-19 datasets and identify several issues with the reliability of external validation testing. The authors therefore present several recommendations for how they

think external validation should be performed, which are essentially that external validation should be done with at least one external dataset and before using a dataset the data similarity with the training dataset should be compared to see if the data is too similar.

The ML test score: A rubric for ML production readiness and technical debt reduction: Breck et al. [64] presents a list of test cases, based mostly on assertions, that they believe should be executed for every model to prove its robustness and fairness. These tests are split up among the testable phases of a ML product. There are tests for the input data, for example to verify feature expectations in the data by encoding them in a schema. For the model itself there are tests like to verify that the offline proxy metrics actually match up with the impact metrics from online usage or to make sure that the hyperparameters have been tuned properly. For what the authors refer to as infrastructure, and what we call learning program and framework, they for example propose unit testing the model code and to verify that it can quickly be rolled back to a previous version in case issues arise. Finally the also propose tests for monitoring the ML system, for example to verify that the developers are notified when dependencies of the models have been updated or to make sure that a model does not get "stale", not being used for a extended period of time, by having it be re-trained regularly. Based on all these tests the authors then propose a test score, where manually doing a test yields half a point and having automated the test yields one point. Each of the sections Data, Model, Infra, and Monitor are summed separately, and then taking the lowest of the scores as the final score.

Themis: Automatically testing software for discrimination: A technique for automatically generating fairness tests, for both causal and group discrimination exhibited by software systems, is Themis by Angell et al. [14]. Group discrimination is defined as the maximum observed difference between the outputs of any two classes of a protected attribute. Causal discrimination is defined as the number of equivalence classes that contain two or more input values, only varying in protected attributes, that produce different outputs from the system under test (SUT). Themis works by defining equivalence classes on the input space of the SUT, and then several optimisations are utilised to minimise the test suite size. It continually generates new tests (i.e., combinations of unprotected attributes) until the confidence on the statistical error meets the user's specified threshold. The authors claim that Themis is the first automated test generator of its kind and purpose, and they deem it appropriate as a comparative baseline for future research.

Tools and Practices for Responsible AI Engineering: Yet another toolkit for testing robustness in addition to explainability has been proposed by Soklaski et al. [65], called rAI-Toolkit (short for responsible AI-Toolkit). It is a software library written in Python that partly offers functionality for assessing the robustness of a model. The authors demonstrate an application of the toolkit on an image classifier based on PyTorch [66]. Although PyTorch is used to fit neural network models, rAI-Toolkit seems generalisable to traditional ML algorithms, based on its mathematical foundation described in the paper.

Towards a holistic software systems engineering approach for dependable autonomous systems: In the paper by Aniculaesei et al. [67] the authors focus on the concept of dependability, which they break down into security, privacy, and safety. To achieve these dependability goals the authors propose to use dependability cages that can be created from existing software artefacts. These cages are designed to be able to deal with both external and internal problems for the software. They work by assuming that the tested behaviour of the system and environment are correct, and that any changes to these might lead to issues. So if the internal behaviour changes then that would constitute an internal issue, while if the inputs from the environment around the system change a

lot then that would mean that there might be an external issue if the system has not been built to handle those inputs.

Towards effective metamorphic testing by algorithm stability for linear classification programs: Another work that aims to remedy the issue of the oracle problem is the paper by Yang et al. [68]. Their aim is to help improve the quality of ML programs and also potentially make it easier for developers to identify bugs. They focus on linear classification algorithms such as SVM, liner regression, and linear discriminant analysis. According to the authors the have split MRs into to groups, Past-execution Dependent MR (PD-MR), which is more effective, and Past-execution Independent MR (PI-MR). PI-MR has been the traditional MR approach where the output only depends on the input, but the authors of the paper are proposing PD-MRs that will depend on both the output of a previous run and the input. They propose two PD-MRs in total based on algorithmic stability, with the first one being about replacement of data and the second about removal of data. These MRs are compared with MRs from previous research in the field, and through mutation testing they are according to the authors proven to be more efficient at locating bugs.

Unit tests for stochastic optimization: Schaul et al. [69] proposes a framework for unit test for testing the optimisation algorithms used within the learning programs for training ML models. The authors state that these unit tests are not enough to prove that a model will be good, but it is a must for the program to pass all of them if it is meant to be robust and general. It is therefore meant to be an addition upon the testing of ML models that is already being done. Some examples of what the unit tests are designed to test are metrics such as curvature scales and various noise conditions.

"Why should i trust you?" Explaining the predictions of any classifier: The tool LIME is proposed in [70] as a means for explaining the predictions of a model. LIME is model-agnostic, meaning that it can be applied to any type of machine learning classification model, for example both neural networks and random forest. It works by fitting a more interpretable model, like a decision tree, in the proximity of the input that was predicted. The authors demonstrate its applicability to textual and image data. It explains the predictions of image classifiers by highlighting which parts of the image were most important for the predicted label.

References

[1] Paul Lou Benedick, Jérémy Robert, and Yves Le Traon. "A systematic approach for evaluating artificial intelligence models in industrial settings". *Sensors* 21 (18 Sept. 2021). ISSN: 14248220. DOI: 10.3390/s21186195.

[2] Xiaoyuan Xie et al. "Application of metamorphic testing to supervised classifiers". *2009 Ninth International Conference on Quality Software*. IEEE. 2009, pp. 135–144.

[3] Jonathan Guichard et al. "Assessing the robustness of conversational agents using paraphrases". *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2019, pp. 55–62.

[4] Ezekiel Soremekun, Sakshi Sunil Udeshi, and Sudipta Chattopadhyay. "Astraea: Grammar-based fairness testing". *IEEE Transactions on Software Engineering* (2022).

[5] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. "Automated directed fairness testing". Association for Computing Machinery, Inc, Sept. 2018, pp. 98–108. ISBN: 9781450359375. DOI: 10.1145/3238147.3238165.

[6] Arnab Sharma and Heike Wehrheim. "Automatic fairness testing of machine learning models". *IFIP International Conference on Testing Software and Systems*. Springer. 2020, pp. 255–271.

[7] Christian Murphy, Kuang Shen, and Gail Kaiser. "Automatic system testing of programs without test oracles". *Proceedings of the eighteenth international symposium on Software testing and analysis*. 2009, pp. 189–200.

[8] Zeyu Sun et al. "Automatic testing and improvement of machine translation". *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 974–985.

[9] Junjie Zhu, Teng Long, and Atif Memon. "Automatically Authoring Regression Tests for Machine-Learning Based Systems". IEEE Computer Society, May 2021, pp. 374–383. ISBN: 9780738146690. DOI: 10.1109/ICSE-SEIP52600.2021.00049.

[10] Samuel Ackerman et al. "Automatically detecting data drift in machine learning classifiers". *ArXiv* (2021).

[11] Sebastian Schelter et al. "Automating large-scale data quality verification". Vol. 11. Association for Computing Machinery, 2018, pp. 1781–1794. DOI: 10.14778/3229863.3229867.

[12] Marco Tulio Ribeiro et al. "Beyond accuracy: Behavioral testing of NLP models with CheckList". *arXiv preprint arXiv:2005.04118* (2020).

[13] Aniya Aggarwal et al. "Black box fairness testing of machine learning models". Association for Computing Machinery, Inc, Aug. 2019, pp. 625–635. ISBN: 9781450355728. DOI: 10.1145/3338906.3338937.

[14] Rico Angell et al. “Themis: Automatically testing software for discrimination”. Association for Computing Machinery, Inc, Oct. 2018, pp. 871–875. ISBN: 9781450355735. DOI: 10.1145/3236024.3264590.

[15] Sanjay Krishnan et al. “Boostclean: Automated error detection and repair for machine learning”. *arXiv preprint arXiv:1711.01299* (2017).

[16] Guy Barash et al. “Bridging the gap between ML solutions and their business requirements using feature interactions”. *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 1048–1058.

[17] Joymallya Chakraborty et al. “Can We Achieve Fairness Using Semi-Supervised Learning?” *arXiv preprint arXiv:2111.02038* (2021).

[18] Josip Bozic, Oliver A Tazl, and Franz Wotawa. “Chatbot testing using AI planning”. *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2019, pp. 37–44.

[19] Vânia De Oliveira Neves, Márcio Eduardo Delamaro, and Paulo Cesar Masiero. “Combination and mutation strategies to support test data generation in the context of autonomous vehicles”. Vol. 8. Inderscience Publishers, 2016, pp. 464–482. DOI: 10.1504/IJES.2016.080388.

[20] Erin Lanus et al. “Combinatorial testing metrics for machine learning”. *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2021, pp. 81–84.

[21] Muhammad Hilmi Asyrofi et al. “Crossasr: Efficient differential testing of automatic speech recognition via text-to-speech”. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 640–650.

[22] Diptikalyan Saha, Aniya Aggarwal, and Sandeep Hans. “Data Synthesis for Testing Black-Box Machine Learning Models”. *5th Joint International Conference on Data Science & Management of Data (9th ACM IKDD CODS and 27th COMAD)*. 2022, pp. 110–114.

[23] Eric Breck et al. “Data Validation for Machine Learning”. 2019, pp. 334–347. URL: https://proceedings.mlsys.org/paper/2019/file/5878a7ab84fb43402106c575658472fa-Paper.pdf.

[24] “Datamorphic testing: A method for testing intelligent applications”. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 149–156. ISBN: 9781728104928. DOI: 10.1109/AITest.2019.00018.

[25] Shin Nakajima and Hai Ngoc Bui. “Dataset coverage for testing machine learning computer programs”. *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2016, pp. 297–304.

[26] Aws Albarghouthi and Samuel Vinitsky. “Fairness-aware programming”. *Proceedings of the Conference on Fairness, Accountability, and Transparency*. 2019, pp. 211–219.

[27] Florian Tramer et al. "Fairtest: Discovering unwarranted associations in data-driven applications". *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2017, pp. 401–416.

[28] Shin Nakajima. "Generalized oracle for testing machine learning computer programs". *International Conference on Software Engineering and Formal Methods*. Springer. 2017, pp. 174–179.

[29] Yasasa Abeysirigoonawardena, Florian Shkurti, and Gregory Dudek. "Generating adversarial driving scenarios in high-fidelity simulators". *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8271–8277.

[30] 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.

[31] Tianlu Wang, Diyi Yang, and Xuezhi Wang. "Identifying and mitigating spurious correlations for improving robustness in nlp models". *arXiv preprint arXiv:2110.07736* (2021).

[32] Anurag Dwarakanath et al. "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing". *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2018, pp. 118–128.

[33] Max Bartolo et al. "Improving question answering model robustness with synthetic adversarial data generation". *arXiv preprint arXiv:2104.08678* (2021).

[34] Arvind Ramanathan et al. "Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision". *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2016, pp. 786–791.

[35] Shiqing Ma et al. "LAMP: data provenance for graph based machine learning algorithms through derivative computation". *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017, pp. 786–797.

[36] Yanai Elazar et al. "Measuring and improving consistency in pretrained language models". *Transactions of the Association for Computational Linguistics* 9 (2021), pp. 1012–1031.

[37] Hongjing Guo, Chuanqi Tao, and Zhiqiu Huang. "Metamorphic Testing for Plant Identification Mobile Applications Based on Test Contexts". *International Conference on Mobile Computing, Applications, and Services*. Springer. 2020, pp. 209–223.

[38] Morteza Pourreza Shahri et al. "Metamorphic testing for quality assurance of protein function prediction tools". Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 140–148. ISBN: 9781728104928. DOI: 10.1109/AITest.2019.00017.

[39] Xiaoyuan Xie et al. "METTLE: A metamorphic testing approach to assessing and validating unsupervised machine learning systems". *IEEE Transactions on Reliability* 69.4 (2020), pp. 1293–1322.

[40] Siwakorn Srisakaokul et al. "Multiple-implementation testing of supervised learning software". *Workshops at the thirty-second AAAI conference on artificial intelligence*. 2018.

[41] Sebastião Santos et al. "On Using Decision Tree Coverage Criteria for Testing Machine Learning Models". Association for Computing Machinery, Sept. 2021, pp. 1–9. ISBN: 9781450385039. DOI: 10.1145/3482909.3482911.

[42] Vinodkumar Prabhakaran, Ben Hutchinson, and Margaret Mitchell. "Perturbation Sensitivity Analysis to Detect Unintended Model Biases". *arXiv* (Oct. 2019). URL: http://arxiv.org/abs/1910.04210.

[43] Christian Murphy, Gail E Kaiser, and Lifeng Hu. "Properties of machine learning applications for use in metamorphic testing" (2008).

[44] Thorsten Joachims. "Making large-scale svm learning practical. advances in kernel methods-support vector learning". *http://svmlight.joachims.org/* (1999).

[45] Ke Wang and Salvatore J Stolfo. "Anomalous payload-based network intrusion detection". *International workshop on recent advances in intrusion detection*. Springer. 2004, pp. 203–222.

[46] Philip Gross et al. "Predicting electricity distribution feeder failures using machine learning susceptibility analysis". *AAAI*. 2006, pp. 1705–1711.

[47] Anna Saranti et al. "Property-based testing for parameter learning of probabilistic graphical models". *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer. 2020, pp. 499–515.

[48] Karan Goel et al. "Robustness gym: Unifying the nlp evaluation landscape". *arXiv preprint arXiv:2101.04840* (2021).

[49] Jiexi Liu et al. "Robustness testing of language understanding in task-oriented dialog". *arXiv preprint arXiv:2012.15262* (2020).

[50] Steffen Herbold and Tobias Haar. "Smoke testing for machine learning: simple tests to discover severe bugs". *Empirical Software Engineering* 27 (2 Mar. 2022). ISSN: 1382-3256. DOI: 10.1007/s10664-021-10073-7.

[51] Rongjie Yan et al. "Stability evaluation for text localization systems via metamorphic testing". *Journal of Systems and Software* 181 (2021), p. 111040.

[52] Xin Yin et al. "Statistically rigorous testing of clustering implementations". Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 91–98. ISBN: 9781728104928. DOI: 10.1109/AITest.2019.000-1.

[53] Saikat Dutta et al. "Storm: program reduction for testing and debugging probabilistic programming systems". *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019, pp. 729–739.

[54] Yi Qin et al. "SynEva: Evaluating ml programs by mirror program synthesis". *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE. 2018, pp. 171–182.

[55] Wei Yang and Tao Xie. "Telemade: A Testing Framework for Learning-Based Malware Detection Systems". *The Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence* (2018). URL: www.aaai.org.

[56] Kexin Pei et al. "DeepXplore: Automated Whitebox Testing of Deep Learning Systems". Association for Computing Machinery, 2017, pp. 1–18. ISBN: 9781450350853. DOI: 10.1145/3132747.3132785. URL: https://doi.org/10.1145/3132747.3132785.

[57] Diogo Moreira, Ana Paula Furtado, and Sidney Nogueira. "Testing acoustic scene classifiers using Metamorphic Relations". *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2020, pp. 47–54.

[58] Xiaoyuan Xie et al. "Testing and validating machine learning classifiers by metamorphic testing". Vol. 84. Apr. 2011, pp. 544–558. DOI: 10.1016/j.jss.2010.11.920.

[59] Aniya Aggarwal et al. "Testing Framework for Black-box AI Models". IEEE Computer Society, May 2021, pp. 81–84. ISBN: 9781665412193. DOI: 10.1109/ICSE-Companion52605.2021.00041.

[60] Arnab Sharma and Heike Wehrheim. "Testing machine learning algorithms for balanced data usage". Institute of Electrical and Electronics Engineers Inc., Apr. 2019, pp. 125–135. ISBN: 9781728117355. DOI: 10.1109/ICST.2019.00022.

[61] 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.

[62] Nick Hynes, D Sculley, and Michael Terry. "The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets". *NIPS MLSys Workshop* (Vol. 1) (2017). URL: https://github.com/brain-research/data-linter.

[63] Federico Cabitza et al. "The importance of being external. methodological insights for the external validation of machine learning models in medicine". *Computer Methods and Programs in Biomedicine* 208 (2021), p. 106288.

[64] Eric Breck et al. "The ML test score: A rubric for ML production readiness and technical debt reduction". *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 1123–1132.

[65] Ryan Soklaski et al. "Tools and Practices for Responsible AI Engineering". *arXiv preprint arXiv:2201.05647* (2022).

[66] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[67] Adina Aniculaesei et al. “Towards a holistic software systems engineering approach for dependable autonomous systems”. *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*. 2018, pp. 23–30.

[68] Yingzhuo Yang et al. “Towards effective metamorphic testing by algorithm stability for linear classification programs”. *Journal of Systems and Software* 180 (2021), p. 111012.

[69] Tom Schaul, Ioannis Antonoglou, and David Silver. “Unit tests for stochastic optimization”. *arXiv preprint arXiv:1312.6055* (2013).

[70] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “”Why should i trust you?” Explaining the predictions of any classifier”. Vol. 13-17-August-2016. Association for Computing Machinery, Aug. 2016, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.