SOFTWARE TESTING: ISSUES AND CHALLENGES OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Kishore Sugali, Chris Sprunger and Venkata N Inukollu

Department of Computer Science and, Purdue University, Fort Wayne, USA

ABSTRACT

The history of Artificial Intelligence and Machine Learning dates back to 1950's. In recent years, there has been an increase in popularity for applications that implement AI and ML technology. As with traditional development, software testing is a critical component of an efficient AI/ML application. However, the approach to development methodology used in AI/ML varies significantly from traditional development. Owing to these variations, numerous software testing challenges occur. This paper aims to recognize and to explain some of the biggest challenges that software testers face in dealing with AI/ML applications. For future research, this study has key implications. Each of the challenges outlined in this paper is ideal for further investigation and has great potential to shed light on the way to more productive software testing strategies and methodologies that can be applied to AI/ML applications.

KEYWORDS

Artificial Intelligence (AI), Machine Learning (ML), Software Testing

1. INTRODUCTION

1.1. Overview of Artificial Intelligence and Machine Learning.

Artificial Intelligence (AI), a widely recognized branch of Computer Science , refers to any smart machine that exhibits human like intelligence. By learning to emulate the perception, logical and rational thought, and decision-making capabilities of human intelligence, AI helps machines to imitate intelligent human behavior. AI has significantly increased efficiencies in multiple fields in recent years and is gaining popularity in the use of computers to decipher otherwise unresolvable problems and exceed the efficiency of existing computer systems. [2]. In [3], Derek Partridge shows various major classes of the association between artificial intelligence (AI) and software engineering (SE). These areas of communication are software support environments; AI tools and techniques in standard software; and the use of standard software technology in AI systems. Mark Kandel and Bunke, H, in [7], have attempted to correlate AI and software engineering at certain levels and addressed whether or not AI can be directly applied to SE problems, and whether or not SE Processes are capable of taking advantage of AI techniques.

1.2. How AI Impacts Software Testing

Research demonstrates that software testing utilizes enterprise resources and adds no functionality to the application. When regression testing discloses a new error introduced by a revision code, new regression cycle begins. Many software applications often require engineers to write testing scripts, and their skills must be equal to the developers who develop the original

DOI: 10.5121/ijaia.2021.12107

app. This additional overhead expense in the quality assurance process is consistent with the growing complexity of software products; [3]

In order to improve the quality of AI products and to reduce costs, automated testing engineers must continually focus on AI ability, performance, and speed. New applications progressively provide AI functionality, sparing the human testers the challenge to comprehensively evaluate the entire product. Logically, AI would be increasingly needed to certify intelligence- containing systems, in part because the spectrum of possibilities for input and output possibilities is so wide. The aim of this paper is to discuss the issues and challenges that AI methods are facing in the field of software testing. Different AI techniques, such as classification and clustering algorithms, are currently focused primarily on monotonous data to train models to predict precise results [10]. In the case of automated software testing procedures, standard machine learning (ML) and more specifically deep learning methods like neural networks, support vector machines (SVM), reinforcement learning techniques, and decision networks can be trained with input data, combined with the respective output of the application under test.

2. ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

2.1. Types of Machine Learning Algorithms

Based on their function, Machine Learning Algorithms have been divided into various categories. The following are the key groups : [20]

- Supervised learning – ML tries to model relationships and dependencies between the target prediction output and the input features. Input data is called training data and has a known label or result. Algorithms include - Nearest Neighbor, Naive Bayes, Decision Trees, Linear Regression, Support Vector Machines (SVM), Neural Networks.
- Unsupervised Learning - Input data are not labeled and do not have a known result. Mainly used in pattern detection and descriptive modeling. Algorithms includes - k-means clustering and association rules.
- Semi-supervised Learning - In the previous two types, either there are no labels for all the observations in the dataset or labels are present for all the observations. Semi-supervised learning falls in between the two. Input data is a mixture of labeled and unlabeled.
- Reinforcement Learning - It allows machines and software agents to automatically determine the ideal behavior within a specific context, to maximize its performance. Algorithms include Q-Learning, Temporal Difference (TD), and Deep Adversarial Networks.

2.2. Role of AI in Software Testing

The application of methods and techniques of artificial intelligence in software engineering and testing is a progressive research field that has been documented in several completed works. Varieties of AI tools are used to generate test data, research data suitability, optimization and analysis of the coverage and test management. In [8], Annealing Genetic Algorithm (AGA) and Restricted Genetic Algorithm (RGA), the two modified versions of Genetic Algorithm (GA) have been used for testing. ACO can be used for model-based test data generation and groups of ants can effectively explore the graph and generate optimal test data to achieve the test coverage requirement [9]. Test sequence generation has been used for state-based software testing [9]. Many other AI based techniques and methodologies have been used for software testing purposes. Requirements-based testing based on particle swarm optimization (PSO), partition testing also based on PSO, using PSO and GA in the context of evolutionary and structural testing, test case minimization using an artificial neural network (ANN), test case minimization using a data mining info-fuzzy network, building a Bayesian network (BN) model to predict software fault content and fault proneness, use of artificial neural networks (ANN) and info-fuzzy networks (IFN) for test case selection and automated test oracle development, using a neural network for pruning a test cases, software test data generation using ACO and automatic partial order planner for test case generation are just a few of the available literature in this respect.

2.3. Role of AI in Test Case Generation

There has been an increasing interest in the use of AI for application testing and some research has been done on how to manage application testing with the help of AI. The various forms of this technique can be found in a simple query to the ACM library on this subject. Some of these techniques include methods for generating a model-based application, model-based tests, and automating the generation of test cases to make it possible to regenerate the tests each time the application changes and to build automated oracles that model the behavior of the user interface. There have also been cases of generating tests based on artificial intelligence (AI) planning techniques and genetic modeling. It would appear that the implementation of genetic algorithms is an interesting way of approaching the automation of test case improvement [21]. In [22] Intelligent Search Agent (ISA) for optimal test sequence generation, and, Intelligent Test Case Optimization Agent (ITOA) for optimal test case generation were used. Memon [24] has several papers about automatically generating test cases from the GUI using an AI Planner. Similarly in [23], a genetic algorithm is used to evolve new test cases to increase the coverage of test suite coverage while avoiding unworkable sequences.

3. ISSUES AND CHALLENGES OF AI

Owing to the lack of technical expertise and research work results, AI software testing has different challenges. Below, the challenges have been summarized:

3.1. Identifying Test Data

In the field of AI, the model must be trained and tested before being deployed into the real environment. Instead of a software-testing expert, model training and testing is typically carried out by a data scientist or engineer. To conduct model testing, however, a logical or structural understanding of the model is not necessary. Model testing is like the testing conducted at the unit or component level. It is necessary for a tester to understand how data in the model evaluation process has been obtained or defined and used. For example, if production data has been gathered as training data, has this been done at a certain time of day? Is this representative of average use? Is the data set too broad or too narrow? If the sampling is conducted poorly, or if the training data is not representative, the real-world findings will be wrong [15]. How do we use an organized method to prepare quality training datasets and coverage-oriented test datasets for AI-based functional features in learning-based AI for todays' AI systems?

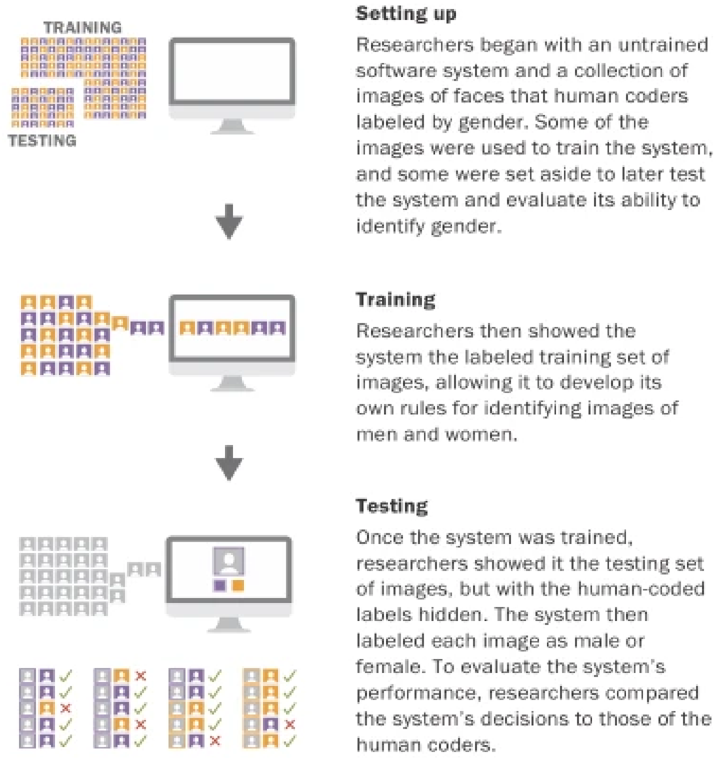
As per [14], Uncertainty in system outputs, responses, or actions has been identified or experienced with AI based mobile apps for the same input test data when context conditions are changed. Similarly, when the training data sets and/or test data sets were amended, accuracy problems were encountered. Moreover, to achieve certain accurate AI model coverage, white-box testing for AI software must pay attention to design training and test data.

In conventional system function testing, we validate whether a system function performs correctly by generating functional test cases based on the specified function requirements in terms of inputs and predicted outputs. In AI software testing, we find that the function correctness is highly reliant on the selection & training of the AI model and the provision of training data sets. For a particular training data set, different AI models typically provide different quality outcomes. For the same AI model, different training and test data sets may generate different outcomes and lead to different correctness. For AI software engineers, this offers a new challenge for quality testing and assurance. Many AI functions, for instance, are designed to accept multiple rich media inputs (text, image, audio, or video events) and produce multiple results in terms of rich media and events/actions. This adds more complexity to the planning of test cases and the generation of test data.

According to [16], Researchers have built a deep learning system to identify gender in images across a variety of human faces, here the challenging aspect leads to adjusting or analyzing images to perform more accurately; and the critical role of the data used to train them in making to perform more (or less) effectively. More than 2,000 unique models were trained and tested based on a common deep learning architecture, and a great deal of variation was discovered in the process in the ability of these models to accurately identify gender in adverse image sets.

The models that were trained using more diverse sets of images (which include the demographic composition and quality and types of images used in each set) were better at identifying gender in a similarly diverse group of photos than models that were trained on data that are more limited. The model that had been trained on images taken from all seven of the individual collections had the best performance. It accurately identified the correct gender for 87% of the training data images, while the models trained using only one of the individual data collections achieved accuracies between 74% and 82%.

Training a machine learning system to identify men and women in digital images



Setting up

Researchers began with an untrained software system and a collection of images of faces that human coders labeled by gender. Some of the images were used to train the system, and some were set aside to later test the system and evaluate its ability to identify gender.

Training

Researchers then showed the system the labeled training set of images, allowing it to develop its own rules for identifying images of men and women.

Testing

Once the system was trained, researchers showed it the testing set of images, but with the human-coded labels hidden. The system then labeled each image as male or female. To evaluate the system's performance, researchers compared the system's decisions to those of the human coders.

"The Challenges of Using Machine Learning to Identify Gender in Images"

At a broader level, these models appeared to have more difficulty identifying women: Six of the eight (including the model that was built using the most diverse training data possible) were more accurate at identifying men than women. But two of the models were considerably more accurate at identifying women than men and as with their overall accuracy, it is not entirely obvious or predictable why certain models would be better at identifying men than women, or vice versa.

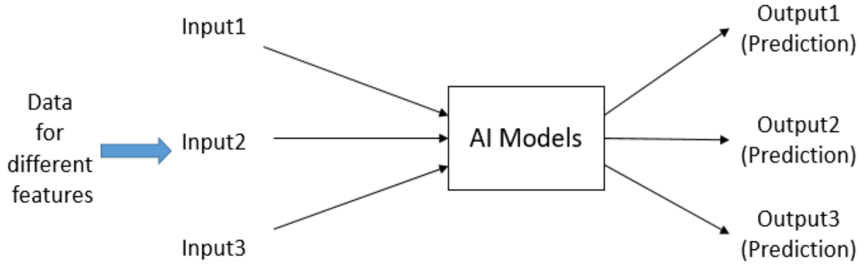
In view of these outcomes, AI models can be inconsistent in identifying accurate results.

3.2. Algorithmic Uncertainty

Uncertainty is a challenge in any kind of software testing. Uncertainty emerges from a non-complete specification or from requirements that are not fully understood. Algorithmic instability is of particular importance in AI applications [1]. This is due to the randomness inherent in the algorithms used in the development of AI. A software engineer designing an AI application does not code to a well-defined algorithm. Instead, the developer is working to train a model with a set of test data. There must be a starting point for neurons and weights in that model. While the developer chooses the number of neurons and the beginning weights; there is no magic formula to determine the correct starting point. The choices made to initialize the model can have a significant impact on how the model evolves during training. The randomness of the resulting algorithm makes many it very challenging for traditional testing strategies. White box test coverage measures become impossible to calculate.

3.3. Lack of Good Measures of Effectiveness

Black-box testing conducts the functionality testing of an application without knowing the structure of implementation. Based on the requirement specifications, test suites for black-box testing are generated. For AI models, applying black box testing will mean testing the model without knowing the features or algorithm used to create the model. The challenge here is to identify the test oracle, which could verify the test output against the expected values in advance [13]. In the case of AI models, there are no expected values in advance. Because of the prediction form of the models, it is not easy to compare or verify the prediction against expected value. During the model development phase, data scientists test the model performance by comparing the predicted values with the actual values. This is not the case with testing the model, where the expected values are unknown for any given input. To assess the effectiveness of the Black box testing on AI models from a quality assurance perspective, we have to find the techniques to test or perform quality control checks. While there are some suggested techniques such as Model Performance Testing, Metamorphic Testing, Testing with various data slices, etc., these seem to be inadequate in the AI environment to evaluate the efficacy of a model.



Challenge is to verify the effectiveness of outputs (Prediction) against the expected value which is unknown. This represents the absence of test oracle

3.4. Lack of a Testable Specification

Intuitively, black box testing techniques should be the most straightforward way to test an AI application because they avoid the intricate inner workings of AI algorithms. One of the most popular black box testing techniques is testing against a requirements specification. However, AI applications present challenges even when testing to a specification. There is a fundamental conflict with the very notion of AI requirements. AI applications are intended to produce generalized behavior, while a requirement is intended to document specific behavior. An AI requirement specification by its nature is then attempting to specify general behavior. Validation of an AI application involves testing the quality of the algorithm's prediction or performance [11]. Measuring the prediction quality in a testable way is very challenging.

In [1], we see examples of typical AI requirements that illustrate the testing challenge. We may have an application designed to carry out facial recognition. The requirement is to recognize a person from a picture. It is one thing to be able to train the model with a collection of photos of a person from one photo shoot with the person in slightly different poses and then to test it on a different set of photos from the same photo shoot. However, this limited capability would certainly not be good enough. There are several more variants that can be required to be handled by the application. What if we add makeup or change the color or length of the hair? What about aging, weight loss, or weight gain? It would not be possible to train the model on every variation. What does good enough really look like?

3.5. Splitting Data Into Training and Testing Sets

Typically, a data scientist will use a framework for automatically splitting the available data into mutually exclusive training and testing datasets. According to [15] one popular framework used to do this is SciKit-Learn, which allows developers to split off the required size of dataset by random selection. When assessing or retraining different models, it is important to override the random seed used to split the data. The results would not be consistent, comparable, or reproducible if this is not done precisely. Typically 70 percent - 80 percent of the knowledge is used for model training, with the remainder reserved for model evaluation. There are numerous specialized methods available to ensure that the splitting of training and testing data has been carried out in a representative manner. It should be calculated in terms of data rather than lines of code when considering the coverage of model testing.

An important point for consideration is the design of regression testing activities. In traditional software development, unless very significant changes are made, the risk of functional regression is typically low. In the case of AI almost any change to the algorithm, model parameters, or training data usually needs the model to be rebuilt from scratch, and the risk of regression for previously tested functionality is very high. This is because rather than a small percentage of change in the model based on the required modifications, 100% of the model could potentially change. In summary:

- The way the initial data was obtained is important to understand whether it is representative.
- It is important that the training data is not used to test the model otherwise testing will only appear to pass.
- The data split for training and testing may need to be made reproducible.

Improper splitting of datasets into training, validation, and testing sets can result in overfitting and underperformance in production.

For instance, if a trained model expects a certain input feature, but if that feature isn't passed to the model at inference time, the model will simply fail to render a prediction. Other times however, the model will fail quietly. Data leakage is one of the most important issues for a data scientist to understand. If you don't know how to prevent it, the leakage will come up often, and it will ruin your models in the most subtle and dangerous ways. In particular, leakage causes a model to look precise before you start making decisions with the model, and then the model becomes very imprecise.

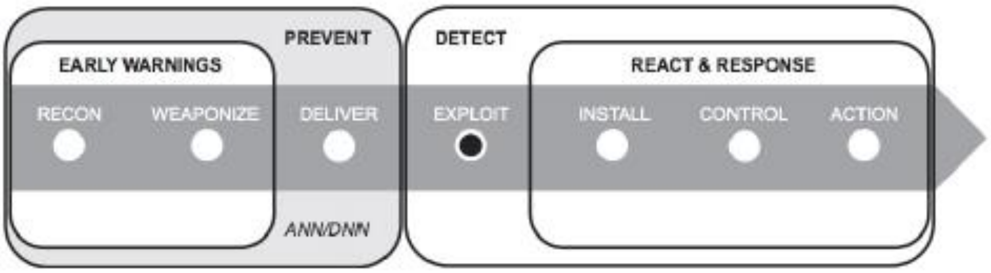
3.6. Incomprehensibility

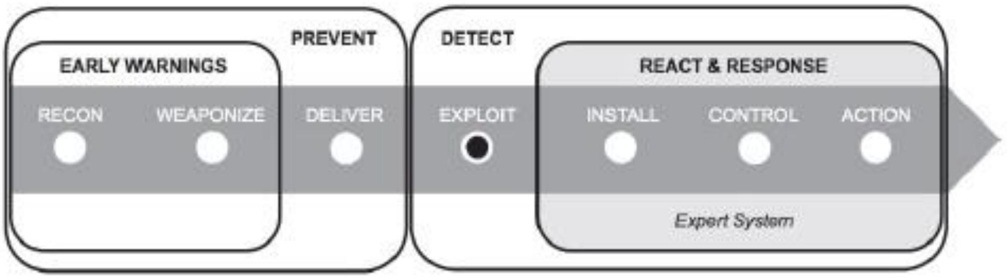
As alluded to in section B above, white box testing techniques are particularly challenging in AI applications. One of the biggest drivers is the concept of incomprehensibility. A neural network by its nature is very difficult to understand logically [1]. This makes many traditional methods of measuring code coverage such as branch or decision coverage next to impossible to compute. Identification of strategies to approximate an equivalent coverage metric for AI applications is an area ripe for further research.

The way a developer creates a neural network varies greatly from traditional software development. In traditional software development, the coder must develop and implement an algorithm for solving a problem. To be able to generate the code correctly in traditional software development, the developer has to understand every detail and every case. This is not the case in neural net development. The goal is to be able to train a model from a collection of test data in order to generalize. Instead of designing a specific algorithm, the coder decides on a number of levels and how many neurons to be included in each level of the neural net. The neural net is then trained on a set of test data. Theoretically, the neural net could be transformed to a complex decision tree of if statements [12]. While a human can be able to read one of the if statements, that does not imply that they could fully comprehend the larger system of if statements in the decision tree. Consider a neural net trained to recognize dogs from pictures. Pictures of several different types of dogs as well as pictures of other types of animals which are not dogs must be included in the training data. After training, you might test the neural net with a series of pictures that were not part of the training data. You might as well begin with a picture of a fish, then a husky, then a wolf, then a poodle. How the net will identify the fish as not a dog and the husky as a dog is not too difficult to grasp. It should then also consider the poodle as a dog, however, and the wolf as not a dog. A wolf looks like a husky a lot more than a poodle does. How would you describe in detail how to recognize the poodle as a dog and the wolf as not a dog? This is only a simplistic instance of where incomprehensibility begins to creep in. There are many more tricky cases.

3.7. AI in Cybersecurity

In today's world, cyber security is the greatest problem. Though cyber security awareness has increased, significant sums of money are being spent and attempts are being made to tackle cybercrimes. Even though several frameworks, software applications, and appropriate steps are taken, cyber risks still emerge from possible malicious actions for different reasons. Organizations, businesses, and researchers are heading towards cybersecurity with Artificial Intelligence to solve cyber problems or challenges. The framework, “Intelligent Cyber Police agents for Early Warnings in an integrated security approach”, is introduced in [17] using Artificial and Deep Neural Networks to prevent attacks along with Expert systems to support reaction and response measures within an integrated security approach, as seen in the figures below.





Despite advantages of the aforementioned proposal to use AI in the cybersecurity domain, the following are the issues and risks of using AI.

1. Data privacy is a major challenge for big data. The analysis of huge volumes of data may cause organizations to be concerned about the privacy of their personal data, and some are even reluctant to share their data. Given that AI methodologies are designed to learn and provide predictive analysis, organizations are concerned about the transparency of their personal data [18].
2. While there are different legal concerns regarding AI, the loss of human control over the consequences of AI's autonomy is the main concern. Due to the unique and unforeseeable nature of AI, existing legal frameworks do not necessarily apply [19].
3. While the adaptation of AI algorithms to cybersecurity has made great strides, they are not yet fully autonomous or can completely replace human decisions. Tasks that need human intervention are still there.

The most common use cases for AI in cybersecurity are listed below, where some evidence of real-world business use has been found:

- AI for Network Threat Identification
- AI Email Monitoring
- AI-based Antivirus Software
- AI-based User Behavior Modeling
- AI for Fighting AI Threats

AI-use can still be defined as nascent in cybersecurity systems. Businesses need to ensure that their systems are trained with feedback from cybersecurity experts, which would make the softwares stronger than conventional cybersecurity systems to detect real cyber threats with much greater precision.

Another challenge for companies using purely AI-based cybersecurity detection methods is to reduce the number of false-positive detections. If the program knows what has been tagged as false positive results, this could theoretically become easier to do. The algorithms will flag statistically significant deviations as anomalies once a baseline of behavior has been constructed and warn security analysts that further investigation is needed.

3.8. Adaptive Behavior

Adaptive behavior is yet another challenge in the testing of AI applications. A primary goal in the development of AI applications is to collect a robust set of clean data from the real work and to use that data to train a model that is generalized to fit that data. This makes the performance of the application dependent on future data matching the historic data on which the model was trained. In many real-world scenarios, the final data population is dynamic [5]. Think of seasonal shopping patterns or disease rates. These shifts in the data cannot be tested if they did not exist in the training data. At best, the historic data can be divided into a training and testing set where the training data set is used to train the model and the testing set is used to test model loss. These tests can show that the model has been generalized and works well with the data available during implementation, but what would be the result when a shift is seen in future real-world data.

It is actually very common for AI techniques to be utilized in scenarios where the target environment is expected to be dynamic [1]. Voice recognition, for instance, is a popular field for AI implementation. Suppose a model was trained using voice samples of native speakers of English from across the United States. Despite regional variations within the US, this application can be trained to run very well. Now, for people who speak English as a second language, imagine beginning to use the same application. Less promising outcomes will possibly begin to be seen in the output of the application. The dynamism of the target environment then allows the AI model to be recalibrated from time to time to account for changes in the new data. Detecting the ideal time to recalibrate the model and even automating the recalibration is an area ripe for further research.

3.9. Communication Failure in Middleware Systems/API Resulting in Disconnected Data Communication and Visualization.

In order to concentrate on the accuracy and completeness of data flows, system integration testing is very important, since much of the software development effort typically takes place in data extraction, normalization, cleaning, and transformation [15]. Any data errors can cause important and complex failures in the quality of the system. For example:

· An error in the transformation logic for birth dates could lead to a large error rate in an algorithm that relies heavily on predictions based on the age of someone.
· A failure to send the correct outcome to the actuators can contribute to system behavior that does not support its goals.

To ensure that all inputs to the algorithm, including from all sensors in the AI environment, have been checked is a challenging job for integration testers. Also intricate is verifying if the number of inputs or records loaded into the device often match the anticipated outcome, ensuring that data enters the system without data loss or truncation, and that after transformation, the data features fulfill expectations.

3.10. Overfitting the model

In AI applications, overfitting is a classic research problem. The fuel required for any AI system is a robust collection of good data. One of the first steps is to divide the data into a data set of training data that will be used to train the model and a data set of testing that will be withheld from training and used to evaluate the AI model instead. When the model knows the training data too well, overfitting occurs. The model begins to understand the noise in the training data and a large difference between the error rate in the training data and the error rate in the test data is observed [4]. The goal when training the model with testing data is for the model to generalize. Often simpler is better. Consider an AI application as an example whose purpose is to predict the existence or absence of a specific disease. In the data for the training of this model, there may be several different variables to consider, but we will use 2 dimensions for simplicity in visualizing. Here, x indicates the presence of the disease and the absence of the disease will be expressed by y.

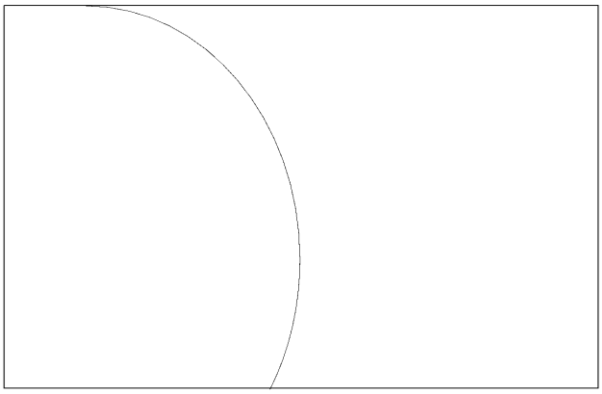


Figure 1 – Generalized model

Figure 1 shows a simple model that is generalized based on the test data. Note that the model is not perfect. The red x and the red 0 represent model loss on the test data. This is acceptable and is to be expected. The only way to account for the two misses would be to complicate the model curve which would result in overfitting.

Figure 2 shows what an overfit model might look like. The equation of the curve is much more complex and now perfectly fits the training data. The issue arises when we start to introduce test data that the model has not yet seen. The two blue question marks represent instances of test data. The model in Figure 2 will predict the top question mark as diseased and the bottom question mark as no disease while the more general model from figure 1 will predict the opposite. Either model could be correct, but it is more likely that the model in Figure 1 will be correct.

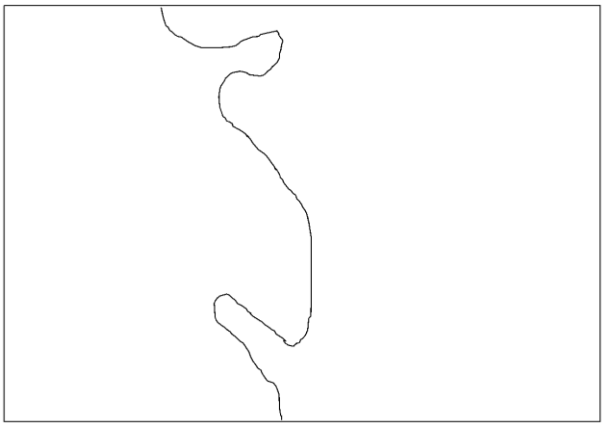


Figure 2 – Overfit model

At first glance, the best approach to building the model may seem to be to reduce losses in training data. However, overfitting results and the model performance on new data degrades. The best model will have an error rate on the training data that is very close to the error rate on the test data.

A common technique to help avoid overfitting is to subdivide the set of training data into multiple subsets called folds. The model is then trained multiple times, once on each fold of training data. This approach provides the model with an opportunity to compare the results from each fold of training data to see what works best and strengthen the generalization of the model. Providing the model with the opportunity to learn from multiple different combinations of data decreases the chances of overfitting [6].

4. Related Work

There has been little research done in terms of software testing in AI. In [14] key challenges of AI software quality validations have been summarized, which include how to define quality assurance standard systems and develop adequate quality test coverage [11], how to connect quality assurance requirements and testing coverage criteria for AI systems based on big data, and how to use systematic methods to develop quality test models. [15] Presents several new challenges that AI systems are facing in predicting system behavior like determining the exact pre-conditions and inputs to obtain an expected result, define expected results and verify the accuracy of test outputs, and measure the test coverage. Also [15] interprets the various challenges in test generation on the code, unit, module, or component levels, and , generating tests from code makes it very challenging for AI to understand the state of the software and its data and necessary dependencies. Parameters may be complex and goals as well as output optimizations may be unclear. The challenges that the model has in adjusting itself in performing more accurately in recognizing gender images [16] has been described clearly. [1] Focused on the issues faced in terms of testing facial recognition in AI systems. Data privacy is another major challenge in AI methodologies because of predictive analysis. Organizations are concerned about the transparency of their personal data [18]. Overfitting is another challenge that AI models are facing today [4] due to the gap in the training data and test data selection. [15] defines the issues with the AI system integration testing in terms of transforming, cleaning, extraction, and normalization of data.

5. Conclusion

In AI/ML applications, software testing is just as critical as it is in any other software development form. Due to the nature of how AI systems work and are built, there are many difficulties that software testers face with AI applications. Such issues range from conventional methodologies for white box and black box testing to problems that are very special to AI systems, such as overfitting and proper splitting of data into training and test data sets. In the efficient testing of AI/ML applications, this paper has enumerated and provided an overview of ten challenges For future studies, each of these challenges is ideal for seeking solutions that alleviate the problem and illuminate the path to more efficient software testing methods and methodologies that can be applied to AI/ML applications.

References

[1] H.Zhuy et al., “Datamorphic Testing: A Methodology for Testing AI Applications”, Technical Report OBU-ECM-AFM-2018-02.

[2] Partridge D. “To add AI, or not to add AI? In Proceedings of Expert Systems, the 8th Annual BCS SGES Technical conference”, pages 3-13, 1988.

[3] Partridge, D. " The relationships of AI to software engineering", Software Engineering and AI (Artificial Intelligence), IEE Colloquium on (Digest No.087), Apr 1992.

[4] S. Salman, X. Liu, "Overfitting Mechanism and Avoidance in Deep Neural Networks", arXiv:1901.06566v1 [cs.LG], 19 Jan 2019.

[5] Enrico Coiera, MB BS, PhD, "The Last Mile: Where Artificial Intelligence Meets Reality", (J Med Internet Res 2019;21(11):e16323) doi 10.2196/16323.

[6] G. Handelman, et. al., "Peering Into the Black Box of Artificial Intelligence: Evaluation Metrics of Machine Learning Methods", doi.org/10.2214/AJR.18.20224, August 2, 2018.

[7] Last, M., Kandel, A., and Bunke, H. 2004 Artificial Intelligence Methods in Software Testing (Series in Machine Perception &Artifical Intelligence "Vol. 56). World Scientific Publishing Co., Inc.

[8] Xie, X., Xu, B., Nie, C., Shi, L., and Xu, L. 2005. Configuration Strategies for Evolutionary Testing. In Proceedings of the 29th Annual international Computer Software and Applications Conference - Volume 02 (July 26 - 28, 2005). COMPSAC. IEEE Computer Society, Washington, DC.

[9] H. Li and C. P. Lam, "Software Test Data Generation using Ant Colony Optimization", appeared in Proc. ICCI 2004, 2004.

[10] Manjunatha Kukkuru, "Testing Imperatives for AI Systems", https://www.infosys.com/insights/ai-automation/testing-imperative-for-ai-systems.html.

[11] L Du Bousquet, M Nakamura, "Improving Testability of Software Systems that Include a Leaning Feature", Learning, 2018 - pdfs.semanticscholar.org.

[12] R.V. Yampolskiy, "Unexplainability and Incomprehensibility of Artificial Intelligence", arXiv preprint arXiv:1907.03869, 2019 - arxiv.org.

[13] A. Kumar, "QA: Blackbox Testing for Machine Learning Models", 4 Sep 2018,https://dzone.com/articles/qa-blackbox-testing-for-machine-learning-models.

[14] J. Gao, et. al., "What is AI testing? and why", 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), DOI: 10.1109/SOSE.2019.00015.

[15] Alliance For Qualification, "AI and Software Testing Foundation Syllabus", 17 September 2019, https://www.gasq.org/files/content/gasq/downloads/certification/A4Q%20AI%20&%20Software%20 Testing/AI_Software_Testing_Syllabus%20(1.0).pdf.

[16] S. Wojcik, E. Remy, "The challenges of using machine learning to identify gender in images", 5 September 2019, https://www.pewresearch.org/internet/2019/09/05/the-challenges-of-using-machine-learning-to-identify-gender-in-images/.

[17] N. Wirkuttis, H. Klein, "Artificial Intelligence in Cybersecurity", Cyber, Intelligence, and Security | Volume 1 | No. 1 | January 2017.

[18] Tom Simonit, "Microsoft and Google Want to Let AI Loose on Our Most Private Data", "MIT Technology Review", April 19, 2016,, https://www.technologyreview.com/s/601294/microsoft-and-google-want-to-let-artificial-intelligence-loose-on-our-most-private-data/.

[19] Bernd Stahl, "Ethical and Legal Issues of the Use of Computational Intelligence Techniques in computer security and Forensics", The 2010 International Joint conference in Neural Networks.

[20] David Fumo, https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[21] Benoit Baudry, "From genetic to bacteriological algorithms for mutation-based testing", Software Testing, Verification & Reliability, pp 73–96, 2005.

[22] V. Mohan, D. Jeya Mala "IntelligenTester -Test Sequence Optimization framework using Multi-Agents", Journal of Computers, June 2008.

[23] Si Huang, Myra Cohen, Atif M. Memon, "Repairing GUI Test Suites Using a Genetic Algorithm" by in ICST 2010: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation, (Washington, DC, USA), 2010.

[24] Memon, A. M., Soffa, M. L. and Pollack, M. E., Coverage criteria for gui testing. ESEC/FSE-9: Proceedings of the 8th European software engineering conference held jointly with 9th.