A4Q

AI and Software Testing

Foundation

Syllabus

Released

Version 1.0

Alliance for Qualification

Version 1.0

Revision History

Version	Date	Remarks
0.2	31 March 2019	First content-complete draft
0.2.5	3 April 2019	Copy edited draft, ready for alpha review
0.3	16 April 2019	Author comments incorporated, ready for beta review
0.3.5	24 May 2019	Incorporated changes.
0.3.6	11 June 2019	Author updates based on beta review.
0.3.7	12 June 2019	Author updates based on beta review.
0.3.8	13 June 2019	Author updates based on beta review.
0.4.0	14 June 2019	Version to handle late beta feedback (second round of beta fixes).
0.5.0	22 June 2019	Proposed final version
0.6.0	6 July 2019	New proposed final version, with the third round of late beta comments processed.
0.6.1	14 July 2019	Minor updates to acknowledge sources of some definitions.
0.6.2	16 July 2019	Minor updates due to a fourth round of beta comments.
0.7	21 July 2019	Copy edit
1.0	17 September 2019	Final copy edit, technical edit, and formatting work for released version.

Table of Contents

0 Introduction

0.1 Purpose of this Syllabus

This syllabus presents the business outcomes, learning objectives, and concepts underlying the AI and Software Testing Foundation training and certification.

0.2 Examinable Learning Objectives and Cognitive Levels of Knowledge

Learning objectives support the business outcomes and are used to create the certified AI and Software Testing Foundation exams.

In general, all contents of this syllabus are examinable at a K1 level, except for the Introduction and Appendices. That is, the candidate may be asked to recognize, remember, or recall a keyword or concept mentioned in any of the three chapters. The knowledge levels of the specific learning objectives are shown at the beginning of each chapter, and classified as follows:

- K1: remember
- K2: understand
- K3: apply

The definitions of all terms listed as keywords just below chapter headings shall be remembered (K1), even if not explicitly mentioned in the learning objectives.

0.3 The AI and Software Testing Foundation Exam

The AI and Software Testing Foundation exam will be based on this syllabus. Answers to exam questions may require the use of material based on more than one section of this syllabus. All sections of the syllabus are examinable, except for the Introduction. Standards, books, and other literature may be included as references, but their content is not examinable, beyond what is summarized in this syllabus itself from such standards, books, and other literature.

The exam shall be comprised of 40 multiple-choice questions. Each correct answer has a value of one point. A score of at least 65% (that is, 26 or more questions answered correctly) is required to pass the exam. The time allowed to take the exam is 60 minutes. If the candidate's native language is not the examination language, the candidate may be allowed an extra 25% (15 minutes) time.

0.4 Accreditation

The A4Q training titled AI and Software Testing Foundation is the only accredited training course for the content presented in this syllabus.

0.5 Level of Detail

The level of detail in this syllabus allows internationally consistent exams. In order to achieve this goal, the syllabus consists of:

- General instructional objectives describing the intention of the Foundation Level
- A list of terms that students must be able to recall
- Learning objectives for each knowledge area, describing the cognitive learning outcome to be achieved
- A description of the key concepts, including references to sources such as accepted literature or standards

The syllabus content is not a description of the entire knowledge area of artificial intelligence and software testing; it reflects the level of detail to be covered in the Foundation Level training course. The Foundation Level course strives to provide a foundation of understanding that enables participation in projects that involve, and adoption of, AI and software testing concepts.

0.6 How this Syllabus is Organized

There are three chapters with examinable content. The top-level heading for each chapter specifies the time for the chapter; timing is not provided below chapter level. For the A4Q AI and Software Testing Foundation training course, the syllabus requires a minimum of 17 hours, 10 minutes of instruction, distributed across the three chapters as follows:

Chapter 1: AI and Software Testing Background 360 minutes

Chapter 2: Testing Artificial Intelligence Systems 400 minutes

Chapter 3: Using AI to Support Testing 270 minutes

0.7 Business Outcomes

AIF-BO-1	As an AI and Software Testing Foundation certified tester, I can perform test analysis, test design, test implementation, test execution, and test completion activities for a system that integrates one (or more) AI-based components.
AIF-BO-2	As an AI and Software Testing Foundation certified tester, I have the ability to AI support testing activities in the context of my organization to facilitate the test process.
AIF-BO-3	As an AI and Software Testing Foundation certified tester, I can contribute to the evaluation of AI-based testing approaches and related tooling in the context of my organization.

0.8 Acronyms

AI	Artificial intelligence
ML	Machine learning

1.0 Key Aspects of Artificial Intelligence

Keywords

Artificial Intelligence, clustering, correlation, decision trees, Deep Learning, expert knowledge-based system, data feature, hyperparameter, machine learning model, regression model, machine learning, neural network, reinforcement learning, sub-symbolic AI, supervised learning, symbolic AI, unsupervised learning

Learning Objectives for Key Aspects of Artificial Intelligence

1.1 What are Human Intelligence and Artificial Intelligence?

AI-1.1.1 (K2) Recognize different types of intelligence modalities according to the theory of multiple intelligence.

AI-1.1.2 (K2) Explain the Turing Test and its limitations.

1.2 History of AI

AI-1.2.1 (K1) Recall the main periods of AI history.

AI-1.2.2 (K2) Explain the difference between symbolic and sub-symbolic AI.

1.3 Symbolic AI

AI-1.3.1 (K2) Discuss the difference between propositional logic, predicate logic and many-valued logic.

AI-1.3.2 (K2) Describe how a knowledge-based system works.

AI-1.3.3 (K2) Explain what a constraint satisfaction problem is.

1.4 Sub-symbolic AI

AI-1.4.1 (K2) Distinguish several types of machine learning.

AI-1.4.2 (K2) Give examples of applications for a given machine learning type.

AI-1.4.3 (K2) Give examples of machine learning algorithms used for a given machine learning type.

AI-1.4.4 (K1) Recall machine learning metrics.

1.5 Some ML Algorithms in More Detail

AI-1.5.1 (K2) Explain how Bayesian belief networks work.

AI-1.5.2 (K3) Use a naïve Bayes classifier for a given problem.

AI-1.5.3 (K2) Explain support vector machine techniques.

AI-1.5.4 (K2) Explain the k-means algorithm.

AI-1.5.5 (K3) Apply a perceptron learning algorithm for a given problem.

1.6 Applications and Limits of AI

AI-1.6.1 (K2) Describe typical activities when applying machine learning.

AI-1.6.2 (K2) Explain possible source of biases in AI systems.

AI-1.6.3 (K1) Identify ethical issues in AI systems.

1.1 What are Human Intelligence and Artificial Intelligence?

Human intelligence can be characterized by a specific ability to develop complex behaviors in changing circumstances. The brain is the central organ that controls most of the activities of the body in a very dynamic way. The brain's dynamism comes from its ability to constantly produce self-organization, which is based on dialogues between neurons and the human body [Vanderah 2018].

For simple and well-defined tasks, the computer's performance far exceeds that of the human brain. The computer is unbeatable for repetitive tasks. When properly programmed (and tested), it achieves very high levels of reliability. When a human being is adapting to new tasks, he/she effectively exploits his/her experience, applies his/her accumulated learning and autonomy and makes use of his/her culture and sensitivity instead of memorizing and executing programs like a robot does.

There are many similarities between artificial intelligence and human intelligence for one simple reason: artificial intelligence aims to mimic human intelligence. The term *artificial intelligence* dates back to the 1950s and refers to the objective of building and programming "intelligent" machines capable of imitating human beings. As early as 1950, Alan Turing, a mathematician working on logic and deduction, proposed a method now known as the Turing Test to evaluate the intelligence of a computer. This characterization of artificial intelligence has also inspired authors of science fiction books, and as early as the 1940s, writer Isaac Asimov formulated the laws of robotics, imagining how intelligent robots could interact with humans.

With the very rapid development of AI technologies, the question of the relationship among humans, intelligent systems and ethical issues has become a central issue.

Types of Intelligence

The theory of multiple intelligences highlights the existence of several types of human intelligence. This theory was first proposed by Howard Gardner in 1983 and has since been expanded.

Gardner defines intelligence as "*an ability or a set of abilities that permits an individual to solve problems or fashion products that are of consequence in a particular cultural setting"* [Gardner 1983]. The theory of multiple intelligence today defines nine types of intelligence modalities [Davis 2011]:

- Musical intelligence refers to the capacity to discern pitch, rhythm, timbre, and tone;
- Bodily/kinesthetic intelligence refers to the movement and ability to manipulate objects and use a variety of physical skills;
- Logical/mathematical intelligence refers to the ability with numbers and figures to carry out complete mathematical operations;
- Visual/spatial intelligence refers to the ability to think in three dimensions, making charts and diagrams, and estimating distances;
- Linguistic intelligence refers to the ability of using words, including a convincing way of speaking or note taking;
- Interpersonal intelligence refers to one's ability with social skills, including having successful long-lasting friendships and understanding other people's moods and needs and behaviors;

- Intrapersonal intelligence refers to one's ability to understand one's self, including being productive when working and making appropriate decisions for oneself;
- Naturalistic intelligence refers to the ability of relating to nature and the natural environment in general, animals, flowers, or plants;
- Existential intelligence is defined as the ability to be sensitive to, or have the capacity for, conceptualizing or tackling deeper and fundamental questions about human existence.

Other modalities of intelligence have been discussed, such as the teaching-pedagogical intelligence, or the sense of humor, but these skills do not fit those proposed in Gardner's theory.

The theory of multiple intelligences highlights the very partial nature of intelligence quotient (IQ) tests, which mainly measures linguistic and logical-mathematical abilities.

Turing Test

One of the fundamental problems with intelligence is that no consensual definition has yet been agreed upon; the one given above is just one of many existing definitions. So defining an artificial intelligence without a definition of intelligence is challenging. Alan Turing in 1950 circumvented this issue when he devised his Turing Test. Although nobody can agree what intelligence is, we all agree that humans are intelligent. So if a machine can't be distinguished from a human, that machine also needs to be intelligent (whatever that means). Because of the technical abilities at the time, the test was devised as a chat.

The Turing Test consists of putting a human being in blind verbal conversation with a computer and another human. If the person who initiates the conversation is not able to say which of his interlocutors is a computer, the computer software can be considered to have successfully passed the test. This implies that the computer and the human will try to have a human talk as humans would. To maintain the simplicity and universality of the test, the conversation is limited to text messages between the participants.

Today, the Turing Test has a historical value because the scenario implemented remains quite simplistic. Instead, AI researchers have been more interested in board games that have a high cognitive level, such as chess and go games. On these games, artificial intelligence is now better than humans, since 1997 with IBM's Deep Blue program for Chess and since 2017 for the game of Go with AlphaGo from Google DeepMind.

1.2 History of AI

Artificial intelligence has an ancient history because humanity was concerned very early on with designing machines that simulate human reasoning and behavior. The myths and legends during prehistoric and antiquity times are full of artificial creatures with consciousness and intelligence [McCorduck 2004]. Artificial intelligence as we understand it today was initiated by classical philosophers, including Gottfried Wilhelm Leibniz with his calculus ratiocinator, who tried to describe the process of human thought as the mechanical manipulation of symbols.

The Dartmouth Conference of 1956 launched artificial intelligence as a specific and independent field of research. The research community has been structured from this period onwards, benefiting from the constant progress in computer processing capabilities, but also from the development of specific theories mimicking the human brain. Animal behavior and other bio-inspired mechanisms such as evolution, are also a source of inspiration for AI researchers.

Main Periods of AI History

Since the 1950s, many directions have been followed by researchers in artificial intelligence to simulate human reasoning. Two directions have been widely investigated.

The first is to formalize reasoning through formal techniques based on different forms of mathematical logic and automatic deduction techniques. This direction is called symbolic because the simulation of human reasoning is based on formalization of knowledge through symbol manipulation.

The second major research direction is to simulate the functioning of the neural network. This direction is called sub-symbolic as opposed to the previous one, because it does not use symbolic reasoning systems. For instance, representing mental phenomena through artificial neural networks is one goal of sub-symbolic AI.

Since the 1950s, the history of AI has been a succession of phases of development and hope for rapid results, followed by forms of disappointment and highlighting boundaries and barriers that may seem insurmountable.

The following periods can be distinguished:

- From the 1940s to the end of the 1950s: the early days of AI research:
    - First work on artificial neural networks (Minsky), as the perceptron proposed in 1958 by Rosenblatt
    - Development of automatic deduction techniques based on mathematical logic
- Over the decades from 1960 to early 1980: development of research work in both symbolic and sub-symbolic AI:
    - Development of many automatic deduction systems applied to different fields (natural language processing, problem solving, planning, etc.)
    - Further work on artificial neural networks
    - Identification of the following limits and difficulties: computational complexity, need for computing power not available at the time, difficulty in formalizing general knowledge
- From the 1980s to the early 2000s: the boom in knowledge-based systems in industry followed by disenchantment and declining industrial investment in AI:
    - Knowledge-based systems, based on the symbolic formalization of cognitive reasoning, had their first successes but faced a wall of complexity
    - Disenchantment with industrial results led to a decrease in industrial investment in AI from the 1990s onwards
    - Mathematical models based on probabilistic reasoning started to be integrated into sub-symbolic AI algorithms leading to many new techniques and research prototypes
- Since the beginning of the 2010s: the success of machine learning techniques has led to a very strong acceleration of investments in AI and a multiplication of applications:
    - The availability of vast quantities of diverse data, the constant improvement of sub-symbolic AI techniques and the ever-increasing availability of computing power, has led

to the great successes of applications of AI techniques to solve complex problems (in robotics, medicine, data analysis and decision-making, games, etc.)

- Large companies and a strong startup ecosystem prioritizing a rapid development of AI technologies and their application in all areas of daily life and business

Difference between Symbolic and Sub-symbolic AI

The symbolic techniques of artificial intelligence mimic human reasoning through the formalization of knowledge and algorithmic procedures of automatic reasoning. These techniques include:

- Automated deduction systems based on classical and non-classical logics
- Knowledge representation and reasoning systems
- Constraint-based solving systems for problem solving

The strengths and limitations of symbolic techniques are to formally represent knowledge and its treatment. In comparison to sub-symbolic AI, the reasoning and its results can be explained, and their validity demonstrated. This is also a limitation because many common problems are simply too complex to be formalized by explicit knowledge (for example, to teach a robot to walk, or to translate a text from one language to another).

Human reasoning and behavior are generally not based on knowledge represented by facts, rules and inferences about that knowledge. The human brain does not function as a knowledge-based system. Perceptions, intuitions and reflexes generally do not obey conscious reasoning. Artificial neural networks, statistical AI, and computational intelligence are examples of sub-symbolic approaches that aim to represent this kind of knowledge. Sub-symbolic methods include:

- Machine learning through statistical learning algorithms and artificial neural networks
- Evolutionary computation algorithms inspired by biological evolution

One of the first limitations of sub-symbolic techniques is the difficulty in explaining the reasons for the results in a way that is understandable to a human being. This is because decisions are not formalized in the form of explicit knowledge. Sub-symbolic AI is not provable in the same way as symbolic AI. Another difficulty comes in the necessary quality and quantity of data to train the learning systems. Very little or poor-quality data will result in either completely unusable AI (i.e., it does not work in satisfactory fashion) or increase the bias in the results obtained with sub-symbolic AI. This is one of the most important quality issues in the field and is covered in detail later in this syllabus.

1.3. Symbolic AI

The techniques of symbolic AI are characterized by an explicit formalization of knowledge and general algorithms for reasoning on this formalized knowledge. Mathematical logic and associated inference systems have played a major role in the development of symbolic AI, but search and optimization algorithms are also used to reason about the formalized knowledge.

Mathematical Logic and Inference

The field of mathematical logic has been developed from the middle of the 19th century, but it has its roots in early philosophy such as Aristotelian logic. Modern mathematical logic, also called formal logic, defines theories that include formal notations and deductive methods applied to these notations. Formal logic is a domain that is one of the foundations of theoretical computer science and symbolic AI. In this syllabus, we consider three systems of formal logic that have influenced the development of symbolic AI techniques:

- **Propositional logic** is a formal language constituted from atomic propositions and logical connectors (no, and, or, implies, is equivalent) with parentheses to create formulas. A truth value is associated with each formula, and a truth table is used to calculate the truth value of a formula for each logic connector. Propositional calculation thus provides an inference system for reasoning on well-formed formulas of propositional logic.
- **Predicate logic**, also called first-order logic, integrates the use of quantified variables over non-logical objects, adding the notion of predicate and quantification (two quantifiers: there exists, for all) on top of propositional logic. Predicate logic is a logical system with a stronger expression power than that of propositional logic, making it possible to formalize general knowledge (e.g., natural language sentences) while remaining a two-value logic ("true" and "false"). An important characteristic of predicate logic is to be generally undecidable. This means that there is no finite procedure for determining the truth value of a predicate formula in all cases. But this limitation does not prevent the predicate calculus from being the basis of many reasoning systems, including natural language analysis, planning and expert systems.
- **Many-valued logic** uses more than the two values of truths ("true" and "false"). These can be three-value logics (for example adding "unknown"), or a finite number of truth values, or an infinite number as in fuzzy logic and probability logic. Thus, fuzzy logic allows us to reason on partial truths, which is not possible with two-value logic.

Formal logic plays a very important role in symbolic AI because it allows us to formalize knowledge and to reason on this knowledge (for example, to deduce new knowledge using inference rules).

Knowledge-based Systems

A knowledge-based system is generally structured in two parts: the knowledge base and the inference engine. This separation brings several advantages in the treatment of knowledge:

- Knowledge can be formalized directly by the domain experts. This knowledge representation is explicit and not hidden in the software code.
- The results of the reasoning can be explained: which part of the knowledge base was used, and which inference technique was used. If the results are incorrect, it is possible to analyze the cause (e.g., by detecting inconsistencies in the knowledge base).
- The inference engine is independent of the knowledge base and can be used in different knowledge base systems.

Formal logic-based representations and automated deduction techniques are generally used for knowledge-based systems. They are successfully used in various fields such as expert systems for diagnosis, intelligent tutoring systems and computer system fault diagnosis.

The limitations and difficulties of implementing knowledge-based systems stem precisely from the explicit nature of knowledge and its formalization. It can be complex even for a domain expert to formalize the knowledge used in his or her activity. When the knowledge base becomes large, maintenance and inconsistency detection problems become equally difficult to solve.

Constraint-based Solving Systems for Problem Solving

Many decision-making problems, such as planning or resource allocation for example, can be represented as a constraint satisfaction problem (CSP). A CSP is generally defined as a system to be solved, including variables, the value domains of these variables and the constraints on these variables. A system solution gives a value assignment proposal for variables that satisfies the constraints. A classic approach to CSPs is to formalize the problem with variables taking values into finite domains and to use search algorithms and constraint propagation to solve the system. Modern constraint solvers can rapidly solve large CSPs involving tens of thousands of variables and constraints.

A simple example of a CSP could be to allocate shifts (a variable being the name of a worker allocated to specific shifts), to thousands of workers (value domains of variables), taking into account vacations (constraints).

Work in the field of constraint solving allows CSPs to be dynamic (constraints can be added or removed dynamically during resolution) and flexible (constraints can be considered more or less imperative integrating a notion of preference or priority).

The techniques of constraint representation and solving thus offer a specific approach to represent knowledge to reason based on this knowledge.

1.4 Sub-symbolic AI

Sub-symbolic AI does not seek to formalize knowledge and reasoning. It identifies what can be found in or learned from existing data to make decisions about unknown situations.

Machine learning (ML) is the main domain of sub-symbolic AI, and on which the rest of this syllabus is focused. It relies heavily on statistical techniques to represent and reason with data using different types of learning.

In the field of ML, *techniques*, *algorithms* and *models* are three different but related terms. Techniques refer to mathematical representations and methods adapted to ML. Algorithms define how these representations are treated. The notion of a model refers to a representation of an algorithm trained with training data.

Data science has been developing rapidly over the past decade due to the availability of large amounts of data, the tremendous increase in storage capacity and data transfer speed and the industry's need to optimize decision-making processes based on data. AI and data science have common intersections, especially since machine learning is one of the methods used in data science to perform data processing, predicting and automating decision-making activities.

Types of Learning

Different approaches can be used in the way a machine learning system is trained. Here are three types commonly used:

- Supervised learning is based on labelled training data; i.e., providing the correspondence between input and output. Once trained on labelled data, the machine learning model is used to predict the results using unknown data.
- Unsupervised learning uses ML algorithms to identify commonalities in the data, without providing labels or classification. The objective is to highlight the structures of a dataset provided to the model, and then automatically classify new data.
- Reinforcement learning uses reward and punishment mechanisms to improve the performance of the learning model. In this case, learning is iterative and interactive with the environment. It consists of an exploration phase to learn, then an exploitation phase to use what has been learned.

Semi-supervised learning combines both supervised and unsupervised learning by using a set of labelled and unlabeled data. This technique reduces the amount of labelled data required for training.

Examples of Applications of Different Types of Learning

The applications of sub-symbolic AI algorithms are diverse and based on a type of learning that is context-specific. For example, to use supervised learning, it is necessary to have properly labelled data, and the use of reinforcement learning is better suited to contexts where there is interaction with the environment. Table 1, below, presents representative applications of sub-symbolic AI classified by machine learning type.

Learning Types	Example of applications
Supervised learning	Image recognition Speech recognition Spam detection
Unsupervised learning	Anomaly/fraud detection Gene clustering Image segmentation
Reinforcement learning	Board and video game AI Robotics Traffic light control

Table 1: Examples of application of sub-symbolic learning types

For some applications, such as the autonomous car, combinations of learning types can be used and integrated into a complex system.

Machine Learning Algorithms

A large number of machine learning algorithms have been developed in recent years. Table 2 below provides examples of commonly used algorithms for different types of learning.

Learning Types	Algorithms	Short description
Supervised learning	Bayes belief networks and naïve Bayes classifiers	Statistical model that represents variables and conditional dependencies via a directed acyclic graph.
	Decision trees	Predictive model representing decisions in the form of a tree based on training data.
	Support-vector machines	Hyperplane extraction from the training data to support classification.
	Artificial neural networks	Large class of representation and algorithms, which historically attempted to mimic the neural networks of the brain.
	Logistic regression	Statistical model that represents the dependencies between variables.
Unsupervised learning	K-means clustering	Given a dataset, it identifies which data points belong to each one of the k clusters.
	Autoencoder	Type of artificial neural network which learns by encoding the inputs and then decoding them to produce outputs comparable to the inputs.
	Principal component analysis	Technique to decrease the number of feature variables.
Reinforcement learning	Q-Learning	Identify the optimal action an agent can take depending of the circumstances.
	Deep reinforcement learning	Use neural networks to learn an optimal action in a reinforcement learning approach.

Table 2: Commonly used machine learning algorithms

As the ML domain is very active, new algorithms are frequently proposed, and combinations of algorithms can also be used. The above table gives some common examples, but other combinations can be used in some contexts or research work.

Deep learning is based on artificial neural networks using several layers of neurons to refine the representation and processing of data. Because of the successes achieved with deep learning in

different fields (e.g., the AlphaGo program mentioned in section 1.1), deep learning has become a branch of ML as such.

Machine Learning Metrics

The performance evaluation of a learning system is usually carried out on the basis of metrics such as (see Google Machine Learning Glossary):

- **Accuracy:** The percentage of correct predictions of a classification model.
- **True Positives:** The percentage of actual positives which are correctly identified.
- **True Negatives:** The percentage of actual negatives which are correctly identified.
- **False Positives:** The percentage of actual positives which are not correctly identified. This is also known as a Type I error.
- **False Negatives:** The percentage of actual negatives which are not correctly identified. This is also known as a Type II error.

These metrics can be used to measure the performance of a ML model and also to drive its improvement.

1.5. Some ML Algorithms in More Detail

The applications on which machine learning techniques are used can vary significantly with regard to the nature of the data available, their quantity and the expected result. A data scientist will need extensive experience with these different algorithms to be able to choose the most appropriate ML techniques, algorithms and libraries for the case to be tackled.

In this section of the syllabus, we focus on the presentation of five different algorithms, chosen for their frequent use in ML and also because they differ in their approach to ML.

Bayesian Belief Networks

Bayesian networks are probabilistic graphical models of knowledge. A Bayesian network represents the conditional dependence between data (for example events) according to probabilistic relationships in a directed acyclic graph. On this graph, the nodes designate the probabilistic variables and the arcs represent the conditional dependencies between the variables.

A Bayesian network is well suited to classify according to the observations made on the data. Through learning with labelled data (i.e., supervised learning), a Bayesian network models the relationships between variables using probability factors. The laws of probability are used to make different inferences, such as inferring probabilities on new data or making prediction. The Bayesian theorem used in inference mechanisms was proposed by Thomas Bayes in the 18th century. It expresses the probability of “A given B” as a function of the probabilities of “B given A” and the probability of A.

Bayesian networks are now a very popular method in ML. There are several reasons for this including the following:

- They can deal with large amounts of data and high-dimensional data, using compact and efficient representations.

- They can use several types of inference methods.
- They are based on solid foundations in the field of probability.

Bayesian networks are used for a wide range of applications including prediction, anomaly detection, diagnostics and decision-making under uncertainty.

Naïve Bayes classifier

The naïve Bayes classifier is a type of simple probabilistic Bayesian classification based on Bayes' theorem with a strong independence hypothesis (called naïve). It is used to perform supervised learning classification on discrete data classes. It is implemented by different algorithms and available in many machine learning libraries available as open source.

Based on training data, the naïve Bayes classifier aggregates the information using conditional probability and an assumption of independence between variables. To classify new data (e.g., a test data of the model), it calculates the probability from the discrete distribution on the different characteristics obtained from the trained model.

An advantage of the naïve Bayes classifier is that it is very fast for classification: probability calculations are not very expensive, and classification is possible even with a small data set.

One disadvantage is that it assumes the independence of the features. This is a strong assumption which is generally not true. Conversely, despite this, naïve Bayes gives often good classification results [Zhang 2004].

Support Vector Machine Algorithm

Support-vector machines (SVMs) are used to classify data using supervised learning. An SVM model is a representation of a set of points (representing the data) in a space, separated into categories.

The purpose of separation is to maximize the average difference between the data in one category and another. This gives a hyperplane where training data appears. Once trained, an SVM model allows new data to be separated into a separate category.

Thus, SVM creates a separation between each class of data by a hyperplane that maximizes the distance between the separation boundary and the class points using an algorithm called maximum-margin hyperplane algorithm proposed by Vladimir N. Vapnik in 1963. Since then, various adaptations have been made (e. g., to perform non-linear regressions) and many implementations of SVM algorithms are available in ML libraries.

SVM algorithms are adapted to classification problems, and they may give good results for the belonging of a data input to a class resulting from learning.

Examples of the use of SVM algorithms include image classification, medical diagnosis or handwriting recognition.

K-means Algorithm

K-means is a classification algorithm used in unsupervised ML. It consists of partitioning the data into *k* clusters, by minimizing the distance of each data to the mean of its cluster. It is therefore an optimization problem for which heuristics have been developed using an iterative refinement technique. This does not guarantee optimality but allows efficient implementations.

K-means requires that the number of targeted clusters be defined initially, which sometimes requires several successive attempts to find the number of clusters best suited to the application.

The results of data classification by the k-means algorithm will depend on the number of clusters defined (the K value), and also on the initialization (i.e., the choice of the initial data to initialize the clusters) and the distance function chosen to calculate the distance between the data.

K-means is a clustering algorithm commonly used in a wide variety of applications such as customer behavioral segmentation (e.g., by purchase history, by activities on a platform), document classification, fraud detection, or image compression.

Artificial Neural Networks: Perceptron Learning Algorithm

Artificial neural networks were initially intended to mimic the functioning of the brain (see section 2.1) but are now considered statistical machine learning techniques. The perceptron is one of the first works on the implementation of artificial neural networks. It was invented by Frank Rosenblatt in 1957, with the aim of creating a perceptron machine for image recognition. The perceptron model has been refined in 1969 by Minsky and Papert [Minsky 1969].

The perceptron learning algorithm is used for supervised ML of binary classifiers, meaning to decide if an input belongs to a specific class or not. A perceptron is a computing unit that takes input and weights and returns 1 if the weighted sum is greater than the threshold and 0 if not. From the labelled input data in two classes, the perceptron learning algorithm learns the weight vector that produces the result 1 or 0 according to the calculation performed. The algorithm iterates on the training data provided as input so that the weighted sum gives the correct result for all the training data. The threshold can be set to zero for example, the weights make sure that for the membership of the result class, the weighted sum will be greater than zero (and therefore the result of the perceptron will be 1) and less than zero (and therefore the result of the perceptron will be 0). The perceptron learning algorithm has been proven to be convergent but may require a large number of iterations (i.e., computation time). Implementation optimization has been proposed and is available in ML software.

The perceptron model is often used in the form of multi-layer networks; i.e., the results of several perceptrons become the input of a perceptron in a subsequent another layer

1.6. Applications and Limits of AI

The development of AI allows many applications that at the same time provide new services but also can raise questions about their implementation and ethical issues.

Activities of Machine Learning

As shown in the previous sections, there is no magic in ML algorithms, but rather computations based on different branches of statistics and/or mathematical optimization. The choice of the type of learning and algorithms to be implemented is therefore a key point for the quality of the result, but the best algorithm applied to poor quality data or badly prepared data will yield poor results. The quality and quantity of the data will directly determine how good a trained ML model will be. This is why ML practitioners spend a large part of their time getting data, cleaning data, and exploring data.

Typically, machine learning development work is conducted as an agile activity, moving through iterations of business understanding, data understanding, data preparation, modelling, and deployment. This is reflected in the Cross-Industry Process Model for Data Mining (CRISP-DM) model:

1. Business understanding is about gaining an understanding of the business requirement and the problem to be solved in data science terms.
2. Data understanding is performed with actual data; it helps to identify data quality issues and requirements for preparation. Some initial insights and hypothesis may be produced at this stage, and data items with the highest correlations to desired predictions will be identified.
3. Data preparation is usually a significant piece of work. Most learning models require that data is converted into floating point (decimal) numbers before the model can understand it. Data quality issues need to be resolved or incomplete records removed.
4. Modelling involves applying different learning models with different parameters and mathematically comparing them.
5. Evaluation is a thorough evaluation of the learning model and the process used to create it. Typically, this involves validation of the results in the business context.
6. Deployment generally involves integrating the model with other software, to provide an end to end system which can be integration, system, and acceptance tested.

Possible Biases in AI Systems

AI-based systems are used to make decisions and predictions, based on collected data and algorithms, and these two components can introduce biases in the results. Note that there are two possible forms of biases: statistical bias due to algorithm or sample data (the system does not accurately reflect reality) or sociological bias (the system does accurately represent the modelled reality as reflected in the data).

Inappropriate and unwarranted biases can be linked to data features such as gender, ethnicity, sexual orientation, income level, or age. Cases of bias in AI-based systems have been reported, for example on recommendation systems for bank lending, or recruitment systems or in judicial monitoring.

Here are some examples of possible source of biases:

- Algorithmic bias can occur when the algorithm is incorrectly configured; e.g., when it overvalues some data compared to others. The hyperparameter tuning of ML algorithms is one way to manage this problem.
- Sample bias can occur when the training data is not fully representative of the data space to which ML is applied.
- Inappropriate bias, like racial or gender bias, may be actual bias that is present in the dataset that was truthfully picked up by the ML approach. In some cases, it can also be reinforced by the algorithm used, for example by overvaluing a more widely present data class.

The ML system must therefore be evaluated against the different biases and then act on the data and algorithm to correct the problem. Later in this syllabus we explore bias in more detail and its effect on the quality of the system. Note that it is not sufficient to exclude inappropriate features

(e.g., race or gender) from the sample data to take care of inappropriate bias. These features may be represented as dependent features.

Ethical Issues in AI Systems

The progress of artificial intelligence techniques highlights ethical issues and how society can control the development of these technologies. The ethical aspects of AI are an important aspect of its development and acceptability to society – cf. [European Commission 2018].

Here are some ethical issues:

- **Explicability and accountability:** How to accept a decision-making system for which the choices made are not transparent or understandable?
- **Fairness and non-discrimination:** How to accept an AI-based decision-making system that incorporates gender or racial biases?
- **Respect for democracy, justice and the rule of law:** Can we let an AI system make decisions that would be in contradiction with the law?
- **Transparency:** Is it acceptable to interact without knowing whether it is a human or an AI-based system?
- **Responsibility:** When an AI system (e.g., a self-driving) car damages property or injures or kills an animal or person, who is responsible, legally and morally?
- **Reinforcement of existing bias:** Often, products and services are designed to meet certain expectations, therefore reinforcing existing biases. For example, cleaning robots are female, whereas robots that aim to protect are male, and both are stereotypically white.
- **Consistency:** Humans are generally considered to be mostly consistent (i.e., not drastically changing their behavior without a major cause for such a change). As humans more and more interact with AI-based systems as they would with humans, they often implicitly depend on that consistency. However, AI-based systems can be very inconsistent (i.e., drastically changing its behavior with no apparent cause).

The consideration of ethical issues is essential for trustworthy AI. Many of these issues remain unsolved and under debate. Further details on how these issues relate to testing is covered in Section 2.1.

2.0 Testing Artificial Intelligence Systems

Keywords

A/B testing, actuators, adversarial actor, agency, automation bias, bias, confusion matrix, deterministic system, discreteness, drift, episodicness, external validity, human in the loop, internal validity metamorphic relations, metamorphic testing, model training, observability, overfitting, probabilistic system, sensors, staticness, statistical significance, training data, underfitting, variance

Learning Objectives for Testing Artificial Intelligence Systems

2.1 General Problems with Testing AI Systems

AI-2.1.1 (K1) Recall the characteristics of non-testable systems and non-deterministic behavior of systems.

AI-2.1.2 (K1) Recall the definition of a probabilistic system.

AI-2.1.3 (K2) Give examples of non-testable systems and non-deterministic systems.

AI-2.1.4 (K1) Recall types of problem that can occur with establishing the test basis for an AI system.

AI-2.1.5 (K1) Describe the types of challenge a weak test basis can create for test planning and execution.

AI-2.1.6 (K1) Recall the No Free Lunch Theorem.

AI-2.1.7 (K2) Explain how a machine learning system can have quality issues as a result of the data chosen to train it.

AI-2.1.8 (K1) Recognize that there is a trade-off between bias and variance in machine learning.

AI-2.1.9 (K2) Describe the concept of drift in machine learning systems and its relationship with testing.

AI-2.1.10 (K1) Recall why ethical considerations may require testing activities.

AI-2.1.11 (K1) Recall the concept of “human-in-the-loop” systems.

AI-2.1.12 (K2) Explain how complacency in the form of automation bias can affect system quality.

AI-2.1.13 (K1) Recognize that adversarial actors can manipulate live AI systems.

2.2 Machine Learning Model Training and Testing

AI-2.2.1 (K2) Summarize the process by which machine learning models are trained and tested by data scientists.

AI-2.2.2 (K1) Recall the difference between model training and traditional unit testing.

AI-2.2.3 (K1) Recall types of defects that can arise in machine learning model implementation.

2.3 AI Test Environments

AI-2.3.1 (K2) Give examples of characteristics of machine learning testing environments, including data labelling processes, model versioning and re-training approaches.

AI-2.3.2 (K2) Give examples of characteristics of intelligent agent environments.

2.4 Strategies to Test AI-based Systems

AI-2.4.1 (K1) Recall the PEAS model for describing AI systems.

AI-2.4.2 (K2) Give examples of types of acceptance criteria for AI systems.

AI-2.4.3 (K2) Explain functional testing strategies for AI systems including metamorphic testing, external validity, A/B testing and the participation of experts.

AI-2.4.4 (K3) Apply a metamorphic approach to writing a functional test case.

AI-2.4.5 (K2) Give examples of test levels appropriate for AI systems.

AI-2.4.6 (K3) Apply an appropriate set of testing levels appropriate for different types of AI system including an expert system, computer vision and a recommender system.

AI-2.4.7 (K1) Recall the relevance of integration testing with AI systems by giving examples of integration defects and their potential impact.

AI-2.4.8 (K2) Explain the importance of acceptance testing with AI systems.

AI-2.4.9 (K2) Summarize challenges comparing the performance of AI to human activities.

AI-2.4.10 (K2) Summarize testing techniques for testing for the presence of unwarranted algorithmic bias.

2.5 Metrics for Testing AI-based Systems

AI-2.5.1 (K2) Describe a confusion matrix.

AI-2.5.2 (K2) Explain how statistical significance relates to testing.

2.1 General Problems with Testing AI Systems

As AI systems become more human, they become less predictable, and there are unique problems and quality characteristics which need to be considered when testing such systems.

Most traditional computer systems and programs can be classified as deterministic. A **deterministic system** is a system which, given the same inputs and initial state, will always produce the same output.

Even in software in which developers have intended to introduce random behavior, the behavior is not truly random. Programming languages use pseudo-random techniques such as using the system time, or user behavior like mouse input, in order to appear to generate random outputs.

Software testing is based on the assumption that there is a test basis and a test oracle. As part of the test analysis process, the test basis and the test oracle must be analyzed to determine what and how to test, and the expected results. It is a prerequisite that there is some mechanism to determine whether a given test has passed or failed, this is known as the oracle assumption. If this cannot be practically achieved, then the system under test is a **non-testable system**.

Systems have been becoming less deterministic for some time as a result of multithreaded architectures, and increased system interconnectedness. However, the use of machine learning as a key driver for AI progress has not only led to a revolution in AI, but also a reduction in the degree of determinism in modern systems.

AI systems present several new challenges to predicting system behavior, including when trying to conduct one or more of the following activities:

- Determine the correct test pre-conditions and inputs in order to obtain an expected result
- Define expected results and verify the correctness of test outputs
- Justify defect reports
- Measure the coverage of tests

Systems can be implemented where it is not theoretically possible to determine the expected behavior of a system. They can also be effectively non-deterministic; i.e., it is not easily practical for a tester to determine the expected behavior. For example, a pseudo-random number generator is theoretically deterministic, but the complexity of creating a test model which used the time in nanoseconds to calculate an expected result means it is not practical. For testing specialists (and for the purpose of this syllabus), systems where it isn't practical to generate expected results are considered effectively non-deterministic.

A **probabilistic system** is one where the occurrence of events cannot be accurately predicted. The behavior of such a system can be described in terms of probability, where a certain degree of error is always attached to the prediction of the behavior of the system. Importantly, it is known as probabilistic because of the difficulty of predicting system events without describing them in terms of probability, not because the system inherently uses statistical probabilities.

Non-deterministic systems, as well as probabilistic systems, can be classified as non-testable systems when using traditional methods.

Software Which Is Written To Calculate an Output for Which the Correct Answer Is Not Known

Software which is written to calculate an output for which the correct answer is not known falls into the category of non-testable systems. When implementing AI systems, there are many cases where the truth (i.e., the desired output of the software) is not fully understood through empirical means.

For example, when using a supervised machine learning approach to classify customers into groups based on a prediction of whether they are likely to buy a specific item in an online store, a likely requirement would be to identify the customers who are likely to buy the item if they are sent a specific email. While it is possible to use historical labelled production data (i.e., supervised learning) and use that to validate the model, there is no guarantee the system will have the same success rate with future customers.

This is an example of a **non-deterministic system** and a **non-testable system.**

Real-world Inputs

In the modern systems landscape, and particularly in the field of AI, there are many examples where systems have real-world sensors, which are receiving unpredictable inputs. This is a type of physical **non-deterministic system.** However, it may not be a **non-testable system,** if the sensors can be manipulated in order to achieve stable outputs.

Self-optimization

Systems which learn based on data they experience, or self-optimize based on inputs and outputs, are **non-deterministic**. While it may be possible to create a model and observe the same inputs and outputs, it is unlikely to be practical to do so. These are **non-testable systems**.

Expert Systems

Detailed requirements and designs are often absent or imprecise for expert knowledge-based systems often associated with symbolic AI, and conversational interfaces. These are often built through an iterative refinement processes and user engagement; it is often the case that any documented, detailed requirements and other test bases are not kept up to date.

These systems are not necessarily non-deterministic or non-testable, but instead can be considered complex and may simply have a poorly defined test basis or lack a test oracle.

Perception of Intelligence

In many cases, the correctness of the behavior of the software is perceived differently by different individual users. An example of this is in conversational interfaces, such as voice recognition and response in smartphones or other devices, where different users have different expectations about the variations of vocabulary that should be supported, and experience different results based on their own choice of words and clarity of speech (in comparison to the dialects and speech the system has been trained with).

These systems are technically both **deterministic** and **testable**, but the variety of possible inputs and expectations of intelligence from end users can make the testing quite complicated. For example, it is challenging to list up front all of requirements up-front for replicating human intelligence. This often leads to a shifting or weak test basis, as intelligence judged differently, by different users.

Model Optimization

Systems which use probability, such as machine learning systems and their learning models, are often carefully optimized. As part of model optimization, the algorithms will be adapted to get the most accurate results. There are trade-offs in this process, and it may be that optimization improves overall results, but causes incorrect results in a small number of cases. These systems are **deterministic** and **probabilistic systems,** and **non-testable** using traditional techniques.

AI System Quality Characteristics

ISO/IEC 25010 establishes characteristics of product quality, including the correctness of functionality and the testability and usability of the system, among many others. While it does not yet cater for artificial intelligence, it can be considered that AI systems have several new and unique quality characteristics, which are:

- **Ability to learn:** The capacity of the system to learn from use for the system itself, or data and events it is exposed to.
- **Ability to generalize:** The ability of the system to apply to different and previously unseen scenarios.
- **Trustworthiness:** The degree to which the system is trusted by stakeholders, for example a health diagnostic.

Bias/Variance Trade Off and the No Free Lunch Theorem

In machine learning implementations, it is crucial to remember that the algorithms work by using observed correlations in training data and assuming they apply equally in other contexts. Usually, this is not actually the case, and this means that it is expected that errors occur. This part of the syllabus outlines concepts relating to a systems ability to generalize and learn.

Referring back to the earlier example with the Titanic dataset, we can see in Figure 1 that the majority of survivors were female. Note that 0 corresponds to “did not survive,” and 1 corresponds to “survived.” This means the model is likely to be suggest that females will survive whereas males would not. This training dataset may be representative of the expected data set in the target use of a model, or it may not. If it is not representative, this is very likely to lead to bias in the model.

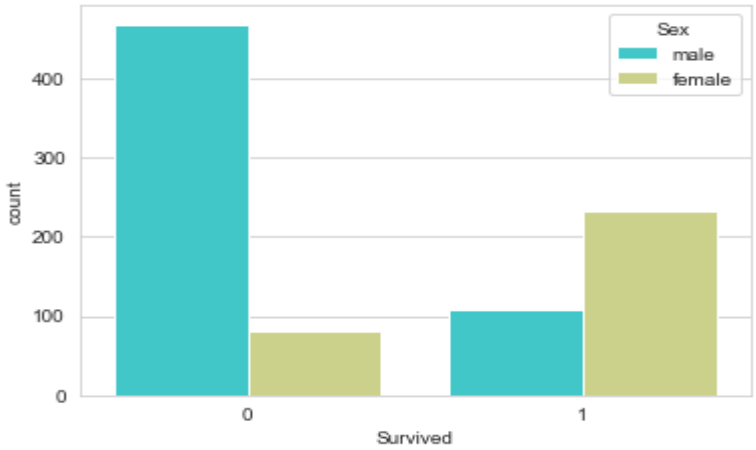

Figure 1: Distribution of gender of Titanic survivors

Because machine learning works by identifying patterns in data and applying them to make prediction, it can be compared to a human being's ability to make decisions based on heuristics or "rules-of-thumb." The concept of equivalence partitioning, which is the assumption that subsets of variables within a similar range will be treated in a similar way, does not always apply. Machine learning algorithms typically consume multiple input data items with complex relationships, and without building an exhaustive understanding of these mathematical relationships, it is necessary to evaluate models using large datasets, to ensure defects are exposed.

Referring again to the Titanic, another passenger ship, the Lusitania, sank three years later. In this example the correlation between survival and gender did not exist, and survival rates were much more closely correlated with age. Therefore, when applying the model we built for the Titanic to data from the Lusitania, we would see a very low accuracy rate.

Bias occurs when a learning model has limited opportunity to make correct predictions because the training data was not representative, and variance refers to the model's sensitivity to specific sets of training data.

Bias is a key statistical concept that is important to understand, that represents the difference between model predictions and reality. Bias occurs whenever the dataset that a model has been trained on is not representative of real data. Bias can be injected without the awareness of the developers of the model; it can even be based on "hidden" variables which are not present in the input domain but are inferred by the algorithm. Bias results from incorrect assumptions in the model, and a high level of bias in the data will lead to "underfitting," where the predictions are not sufficiently accurate.

By contrast, models with a low level of bias suffer from a greater variance based on small fluctuations in the training data, which can lead to "overfitting." For supervised learning, it is an established principle that a model which is trained to give perfect results for a specific dataset will achieve lower quality predictions with previously unseen inputs. Selecting the ideal balance between the two is called the bias/variance trade-off, and this is a property of all supervised machine learning models.

High bias, low variance models will be consistent, but inaccurate on average. High variance, low bias systems are accurate on average, but inconsistent. Examples of low-bias machine learning models include clustering. Examples of high-bias machine learning models include linear regression.

There is a direct correlation between bias and variance, which is why it is referred to as a trade-off. Specifically, increasing the bias will decrease the variance and increasing the variance will decrease the bias.

Another similar area of complexity in machine learning is the concept of the "No Free Lunch Theorem." This is a set of mathematical proofs that show any general-purpose algorithm is of equivalent complexity when its efficiency and accuracy are assessed across all possible problems. Another way of looking at this is to state that all algorithms are accurate to the same degree when averaged across all possible problems.

This is due to the huge potential problem space that a general-purpose algorithm can be applied to. If an algorithm is optimized for a specific sub-domain, it is known that it will be sub-optimal for others. This essentially also means that, when no assumptions at all can be made for a problem (e.g., heuristics to be applied), then a purely random approach will on average perform as well as the most sophisticated ML approach. The reason why the human brain works well in real-life is that our world allows many such assumptions; e.g., that history is a good predictor of the future.

This can be illustrated through the example of a search algorithm of the sort used to identify the route out of a maze. Search algorithms typically enumerate possibilities until they determine a correct answer, and the mechanism of enumeration is critical. For example, a common way to find your way out of a maze is to keep turning in the same direction until you reach an exit, a key variable for an algorithm might be whether to turn left each time or turn right. If the algorithm was configured to always turn left, then given a maze with an exit immediately upon the left, it would perform exceptionally well. If the exit was actually immediately to the right, it would perform very poorly. While this is obviously theoretical, it highlights how a model's effectiveness can vary greatly as the problem space changes.

The problems associated with model optimization are crucial to understand when testing machine learning models. Examples of defects which can occur are listed below:

- Models which are trained with a high variance on a data feature which does not vary much in the training dataset, but varies much more in real life, will be inconsistent in making predictions.
- Models which are trained with a high bias (e.g., models trained on data from a single country but intended to be used globally) may overfit to data items which are not highly correlated with the desired predictions.
- Supervised learning datasets which are incorrectly labelled (e.g., by humans) will lead to models which are incorrectly trained and will make incorrect predictions.
- Integration issues which cause data features to be incorrect or missing can cause incorrect predictions even when the model has been exposed to correct data.

Drift

Drift is the concept that the correlation between the inputs and the outputs of an AI system may change over time. It is not possible to test for drift generally, but it is possible to test specific risks.

When systems are re-trained with the same training data but a new random initialization of a learning model, or self-learn based on new data, then there is a risk that the correlations in the data change and consequently the model's behavior changes. In other words, drift means that the average error rate is unchanged, but that some instances are labeled differently than they were labeled before. These could be hidden correlations that are not necessarily even represented in the input data domain for the AI system. These could also be seasonal or recurring changes, or abrupt changes. They could also be cultural, moral or societal changes external to the system.

It is important to understand that many machine learning models are effectively built from scratch whenever they are re-trained. Traditional software engineering approaches typically change a limited amount of code in each release, whereas in many mathematical based approaches the model is completely rebuilt. This increases the risk of regression significantly.

These correlations can also become less valid over time in the real-world. For example, if the popularity of specific products change, or different groups of users start to use a website more, the predictions and decisions made by machine learning will become less accurate.

Another aspect of the drift in AI systems can be attributed directly to testing. For example, the pesticide paradox referred to in the ISTQB Foundation syllabus states that if the same tests are repeated over and over again, they find less defects. This is also true where machine learning models are used, and automated tests are repeated in a way that changes the model. The model

can in fact become overfitted to the automated tests, reducing the quality in problem spaces that fit outside of the domain of the automated tests.

When testing for drift, it is necessary to either repeat tests over time, similar to regression testing, or by including external data (external validity testing), such as information from the environment that is not available to the algorithm.

Ethical Considerations

An early step in planning a testing activity should include gathering applicable regulatory standards to determine if they impact the testing. Even if stakeholders do not have an explicit regulatory, ethical, or procurement requirement, it is obvious that an error rate in an algorithm which affects a sub-group of people (in an unintended way) can have an adverse effect on quality. This is an important aspect when considering the trustworthiness, to stakeholders, of an AI system.

As demonstrated in the previous section, a model can be built that is highly dependent on personal characteristics (e.g., gender and age), which are not accurate correlations in the real world. If our Titanic model was used currently in the real world, in order to optimize a rescue operation for a sinking passenger ship, consider the implications with regard to ethics.

In some jurisdictions, such as in the EU under the GDPR regulations, there are specific legal requirements that apply where data about actual people is used with complex systems, particularly those which make decisions and can affect people in a significant way (e.g., access to services). These requirements may need:

- Explicit informed consent to automated decision making using the user's data.
- A clear ability to request a human to review the decision.
- Risk assessment mitigated through testing or other activities.

The role of the tester is to identify how ethical considerations can affect product quality and consider appropriate testing strategies. It is not the role of the tester to identify legal requirements that may apply, or make ethical judgements, but they should be aware of the concepts and how to mitigate ethical risks using testing techniques.

Automation Bias

Automation bias, or complacency bias, is the tendency for humans to be more trusting of decisions when they are recommended by AI systems. This can take the form of a human failing to take into account information it receives other than from the system. Alternatively, it can take the form of a failure by the system which is not detected by a human as they are not properly monitoring a system.

An example of this is semi-autonomous vehicles. Cars are becoming increasingly self-driving, but still require a human in the loop to be ready to take over if the car is going to have an accident. If the human gradually becomes trusting of the car's abilities to reason and direct the vehicle, they are likely to pay less attention, and monitor the car less, leading to an inability to react to an unexpected situation.

In predictive systems, it is sometimes common to have a human validate a pre-populated machine recommendation. For example, a procedure where a human keys data into a form might be improved to use machine learning to pre-populate the form, and the human just validates the data.

In these scenarios, it is necessary to understand that the accuracy of the human decision may be compromised, and test for that where possible.

Automation bias is most likely to be detected during acceptance testing.

Adversarial Actors

Hackers and other adversarial actors can attempt to disrupt or manipulate an AI system just like any other modern system. One new attack vector, which is present for systems which learn and self-optimize, is the risk that by manipulating the data inputs to the model, it can be changed. This is particularly relevant for any system which interacts with the real world through natural language, or physical interactions.

Examples can include:

- Manipulating intelligent bots on social media by feeding them inputs which promote a specific viewpoint; e.g., political.
- Moving objects around to make a robot perceive an obstacle where there is not one.
- Making changes to objects in the real-world, which limits or changes the result of sensors.
- Providing incorrect information to an expert system, to change its recommendations.

2.2 Machine Learning Model Training and Testing

It is key to understand the criticality of the training dataset used when determining the quality of the models. While in traditional software, functional flows of input data and actions through algorithms represented by code result in outputs, machine learning is different. A model is created and built using examples of input and (usually) output data, the model then uses statistical methods to determine the correct outputs.

Another type of machine learning is called reinforcement learning, where the mechanism is put in place so that users, or other real world sensors, can tell the algorithm whether it made the correct decision. Examples of this include search engines which monitor which links people actually click on in results and self-improve.

Often, supervised learning models are initiated with a set of human-labelled data. That is input dataset, which includes an additional variable as an expected result. This can often be taken from prior production datasets, but care needs to be taken to ensure the process of collection was itself unbiased and statistically sound. It is not usually necessary to specify a large number of examples; in fact, using too many examples is in itself an example of overfitting.

Throughout the process there are many semi-manual tasks which occur, and many human decisions, including the data items selected for the learning model, the mechanism by which the data is collected, the decision of which algorithm to use, and the integration of data sources. There are plenty of opportunities for human mistakes to inject defects, just like other software projects.

Model training and testing before deployment is normally conducted by a data scientist or engineer rather than by a software testing specialist. However, a mathematical understanding of the model is not required in order to conduct model testing. Model testing is analogous to a form of testing conducted at the unit or component level. It is important for a testing specialist to understand how data has been gathered and used in the model evaluation process. 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 dataset too broad or too narrow? If the sampling is done incorrectly, or if the training data is not representative, the real-world outcomes will be wrong.

Typically, a data scientist would use a framework for automatically splitting the data available into a mutually exclusive training dataset and a testing dataset. One common framework used to do this is SciKit-Learn, which allows developers to split off the required size of dataset through random selection. When evaluating different models, or retraining models, it is important to override the random seed used to split the data. If this is not done, the results will not be consistent, comparable, or reproducible.

Normally 70%-80% of the data is used for training the model, with the remainder reserved for evaluating it. (There are advanced methods available for ensuring that the train/test split has been done in a representative way; however, they are outside the scope of this syllabus.) This is in fact, exactly, what we did with the Titanic example. We used a single line of code to create a random split of data.

Traditional unit testing usually focuses on covering statements and conditions with code. However, a whole machine learning model can often be represented in a very small number of lines of code and it is component tested in isolation. When considering the coverage of model testing, it should be measured in terms of data, rather than line of code.

An important point for consideration is the design of regression testing activities. Whereas in traditional software development the risk of functional regression is usually low unless very large changes are made, almost any change to the algorithm, parameters or training data typically requires rebuilding of the model from scratch, and the risk of regression for previously tested functionality is quite high. This is because rather than changing a small percentage of the model based on the required changes, 100% of the model can potentially change.

In summary:

- The way that the initial data has been gathered is important to understand whether it is representative.
- It is important that the training data is not used to test the model otherwise testing cannot fail.
- The data split for training and testing may need to be made reproducible.

2.3 AI Test Environments

Test environments can be conceptually very different with AI systems in comparison to traditional software environments. They can include physical areas, data collection processes, multiple agents, and many other unusual aspects. It can also sometimes be difficult to distinguish between the AI system and its environment. For instance, is the training data used to train a machine learning model part of the environment, or the system under test? From the context of a testing specialist, and for the purposes of this syllabus, we consider training data and its collection to be part of the test environment, and the model and its parameters part of the system under test.

Most modern artificial intelligence includes some degree of machine learning. In addition to considering the current environment, testing practitioners should be aware of the processes used to collect and/or label data for supervised learning. For example, for supervised learning approaches, organizations may outsource large data sets to be labelled or annotated, in order to provide large

scale training data. Manual annotation of data, for both training and testing sets, should be validated through experimentation and manual review of subject matter experts.

The environment may be continuously self-optimizing or may be restarted to a known model baseline prior to testing. This is an important consideration.

It is important, when considering the machine learning test environment, to consider whether the population of data in use reflects the expected final population. For instance, cultural and societal aspects may vary significantly between different sets of target users. Using data from one part of the world to test a global system can lead to unpredictable results.

It is clear that AI systems to some degree perceive their environment, and to some degree take action to influence it. We can refer to the system's ability to perceive its environment through **sensors**, and its ability to influence the environment through its **actuators**.

Where the AI system is characterized by an impact on the physical, real world, for instance an AI system inside a robot, the test environment may need to be very carefully planned. It may need to consider physical operating parameters such as exposure to weather, in order to fully evaluate it.

One model for describing AI environments is D-SOAKED:

- **Deterministicness.** We have already discussed determinism in the context of an AI system, but is the environment itself deterministic, probabilistic, or non-deterministic? Few AI systems which interact with the real world can claim to have a deterministic environment. For example, if a system is able to make a recommendation to a user searching for something, is the user guaranteed to follow the recommendation? If not, then the environment is either probabilistic, or non-deterministic.
- **Staticness.** Static environments do not change until the AI system takes an action, whereas dynamic environments continue to change.
- **Observability.** A fully observable environment is one where the system has access to all information in the environment relevant to its task.
- **Agency.** If there is at least one other AI system in the environment, it can be described as a multi-agent environment.
- **Knowledge.** The degree to which the system knows how its environment will behave and respond.
- **Episodicness.** Episodic environments comprise series of independent events, whereas sequential environments involve a sequence of events, and behavior will change based on prior events.
- **Discreteness.** A discrete environment has fixed locations, zones or time intervals. A continuous environment can be measured to any degree of precision.

Table 3 uses these characteristics to compare the intended environments for a machine learning model that recommends purchases, and an assistance robot providing support (e.g., to senior citizens or those less able).

	Purchase Recommender	**Assistance Robot**
Deterministicness	Not deterministic	Not deterministic
Staticness	Not static; user may do something else	Not static; the world continues...
Observability	Observable	Partially observable
Agency	Single agent	Multi-agent
Knowledge	High	Low
Episodicness	Episodic	Sequential
Discreteness	Discrete	Continuous

Table 3: Comparison of environment characteristics for two example systems

Note that while these are all valid for the final intended usage, it may be that within a testing context it is practical to make the environment more deterministic, static, observable and discrete.

2.4 Strategies to Test AI-based Systems

Acceptance Criteria

System requirements and design processes are equally important with AI systems, but given the data or rule driven nature of them, often the basis for the detail for how the system should behave is composed differently than a procedural system.

It is very important that the standard of accuracy is agreed up-front between stakeholders. For example, even if only 70% of the system's responses are correct according to an expert, that may be acceptable if the average human falls well short of that.

Ideally requirements for complex AI components should specify both desired and minimum quality requirements and consider false positive and false negative errors. These are discussed further in the metrics section.

Here are some examples of good acceptance criteria:

- A classification algorithm for prediction is desired to achieve no more than 10% false positive errors and no false negative errors (see Metrics), but up to 15%/5% will be acceptable.
- An expert system using a conversational interface is desired to answer all human questions relating to a domain accurately, but 80% is acceptable as long as a human evaluator cannot tell the difference between that system's responses and a human.
- An algorithm that is predicting the right special offer to present to a website visitor should convert to a sale 5% of the time, but as long as it is more accurate than the current algorithm it is acceptable.

One model for describing the desired behavior of AI systems, particularly those that are considered autonomous and able to make decisions based on prior experience, is the PEAS model. This stands for Performance, Environments, Actuators and Sensors:

- Performance in this context is an objective measure of the success of the system's behavior. It can also be phrased as defining how the agent itself knows whether it has been successful. Examples of performance can include speed of execution of an action, or profitability, or safety outcomes. Notes that the definition that related to speed and time is commonly used in other testing contexts.
- Environment is defined in Section 2.3 using the D-SOAKED model.
- Actuators were referred to previously and include anything that allows the system to take action which influences its performance.
- Sensors were also referred to previously and include anything that allows the system to perceive events relevant to its goals.

Functional Testing

This section details five functional testing techniques that can be useful in testing AI systems.

External and Internal Validity

With complex, complicated, or probabilistic systems, it can be sometimes challenging to establish a test oracle. There are a number of techniques which can alleviate this problem. One concept to understand is that of internal versus external validity.

The applicability of external validity is a function of the external data, its diversity, coverage and ease of use. An example of external validity would be taking a system designed to guess someone's age, and consulting census data to determine their actual age.

Metamorphic Testing

Metamorphic testing is a testing technique which requires multiple executions, and uses a test design approach that creates a pseudo-oracle for the system, which is made up of heuristics called metamorphic relations.

These relations are probabilistic in nature and define the likely correlation between multiple inputs and outputs of the system. For example, in a system using machine learning to predict the likelihood a person will have a disease that predominantly affects older people, there is likely to be a strong correlation to age. There is therefore a metamorphic relationship between the input age and the output prediction. A test design could therefore state that in general, a test with a higher input age should generate a higher likelihood output, and vice-versa.

Once metamorphic relations have been established, they can be used in place of the test oracle to generate test cases which verify individual sets of inputs and outputs. In addition, the successful execution of metamorphic tests can lead to the identification of further tests, unlike conventional testing.

In summary, metamorphic testing allows test analysts to predict the results of future tests, based on past test executions. It is usually a kind of internal validity test.

A/B Testing

A/B testing is another kind of testing designed to solve test oracle problems. Conceptually it is a test that is run with two equivalent samples, changing one variable. This kind of testing is heavily used in consumer-facing marketing and e-commerce, where it is not clearly known which kinds of system behavior will lead to the best outcomes (in terms of customer activity).

A simple example of this type of test is where two promotional offers are emailed to a marketing list divided into two sets. Half of the list gets offer A, half gets offer B, and the success of each offer gives insights into the best offer. Many e-commerce and web-based companies use A/B testing in production, diverting buckets of consumers to different functionality, to see which consumers prefer, or achieves the best business outcomes.

This concept can be applied to other situations where the correct result is unknowable, such as recommender and classifier systems. With this concept, an external validity check can be performed on how the system is received in a production environment.

Another useful application of A/B testing is when it is difficult to test the external validity of the system before reaching production. That is, testing is conducted which shows that the system gives the correct results with sample inputs. External validity is often only proven in the wild. By using A/B testing, changes can be released to only a small population of users or use cases, which enables external validity testing to be conducted in a similar way to beta testing.

Evaluation of Real-world Outcomes

Evaluation of real-world outcomes, also known as field testing, is an established approach for testing AI systems. It may be necessary to extend testing into the production environment to provide external validity, in order to utilize knowledge that is beyond the input-output data of the system under test. This may be through A/B testing of new features exploiting user actions and feedback.

It may also be required to evaluate real-world outcomes regularly to prevent drift over time. Understanding the outcomes that stakeholders achieve from the system and comparing back to expected performance from pre-release testing gives an indication of whether models need to be trained and tested again.

Another area where evaluating the real-world outcomes is particularly necessary is when determining the ethical and societal impact of automation. Particularly where hidden variables are affecting the outputs of the system, large amounts of data may be required to get effective results.

Evaluation of real-world outcomes can utilize:

- Internal information about whether the action taken by the system has resulted in the desired goal in the environment
- Structured or unstructured feedback systematically gathered from users
- External data sources that correlate with internally held system data
- Additional sensors fitted to monitor the system or the environment

Expert Panels

When testing rule and knowledge based expert systems, which are aimed at replacing experts themselves, it is clear that human opinion needs to be considered. This is a kind of external validity test, and can be achieved using the following process:

1. Establish a set of test inputs based on analysis of the input domain
2. Perform a Turing Test, using experts in the field of the system under test, to determine the equivalent outputs (likely recommendations) of experts, and therefore the expected results
3. Evaluate the differences between results through a blind rating (again by experts) of the outputs. The purpose of this step is to determine the natural variation in human expert opinion.

In this context, outputs are the results of the system, which could include recommendations or classifications.

There are several considerations which are important while conducting such tests:

- Human experts vary in competence, so the experts involved need to be representative, and the number of experts needs to be statistically significant.
- Experts may not agree with each other, even when presented with the same information.
- Human experts may be biased either for or against automation, and as such the ratings given to outputs should be double-blind (i.e., neither the experts nor the evaluators of the outputs should know which ratings were automated).
- Humans are more likely to caveat responses with phrases like “I’m not sure, but…” If this kind of caveat is not available to the automated solution, this should be considered when comparing the responses.

Levels of Testing

The purpose of different levels of testing for AI systems is to systematically eliminate variables that may lead to quality failures in the system. Table 4 shows examples of testing objectives that could be applied at different test levels, to different types of AI system.

Example	**Component Testing**	**System Integration Testing**	**System Testing**	**Acceptance Testing**
Expert system: helping diagnose problems with your self-driving car	Expert verification of recommendations using virtualized sensors and metamorphic testing.	Verifying that the sensors interact correctly with the expert system.	Verifying that the expert system behaves as expected when fully integrated	Testing with non-expert users in real cars.

Example	Component Testing	System Integration Testing	System Testing	Acceptance Testing
Computer vision: facial recognition and greeting of customers	Model training/ testing based on pictures from files.	Testing that the image from the sensors is of sufficient quality to achieve acceptable results.	Verifying that the recognition system sensors, models and greetings work as expected when fully integrated	Testing with groups of people who look different, to a statistically significant degree.
Recommender: recommending a purchase based on order history	Model training/ testing using prior order history.	Ensure the model is able to retrieve all the inputs to the model correctly.	Verifying that the recommendation system including models and retrieval of order history works as expected when integrated, and make effective recommendations	Conduct A/B testing with some end customers to see if results are improved.

Table 4: Examples of test objectives for each test level with example AI systems

Component Testing

Complex algorithms typically require significant component testing as they contribute significantly to the overall system functionality.

In the context of machine learning models, this is likely to be highly data-driven as described in the Model Training and Testing section.

With expert systems it is common to fully verify the requirements for the knowledge that the system needs to have. As these are usually specified in a machine-readable form, it is often possible to fully automate this testing by using the knowledge inputs as a test oracle.

Systems which take real-world inputs through sensors should have those sensors virtualized, so that they can be controlled and observed.

System Integration Testing

With integration testing, it is very important to focus on the accuracy and completeness of data flows, as the majority of the software development effort is usually in the extraction, normalization, cleaning, and transformation of data. Any data errors can cause significant and complex failures in the quality of the system. For example:

- An error in the transformation logic for dates of birth could cause a large error rate in an algorithm that relied heavily on predictions based on someone's age.
- A failure to send the correct result to the actuators will lead to system behavior which doesn't support its goals.

Integration testing should consider:

- Ensuring that all inputs to the algorithm, including from all sensors in the AI environment, have been verified. Approaches might include:
    - Verifying that the number of inputs or records loaded into the system match the expected result.
    - Ensuring data reaches the system without data loss or truncation, and that the data features meet expectations after transformation.
- Ensuring outbound communications from the system through any actuators, occurs correctly.
- Testing to ensure that sensors/actuators interact with the system correctly, and that the system response to multiple simultaneous inputs is adequate.

System Testing

The goal of many AI systems is to reproduce the intelligence of human experts. If there is a heavily documented set of rules or a pseudo-oracle that the system can be tested against, it may be possible to define expected results which achieve a high level of coverage. However, in many cases there may not be such a mechanism and it may be more appropriate to evaluate system performance against human experts.

It is usually prudent to verify the accuracy of outputs, including the competence of expert recommendations, prior to validating their acceptability to users. Otherwise, it will not be clear whether any failures are caused by inadequate outputs, or human factors.

User Acceptance Testing

Testing with target users is very important if they will be directly interacting with the system. As outlined before the external validity of a system is often not proven until production. However, getting users as involved as possible in the development has many advantages.

Just like with traditional graphical user interfaces, systems with natural language interfaces often find that users use the system in ways that their designers had not anticipated. This may take the form of specific phrases or words that users decide to use to express their needs.

Given the problems with achieving accuracy in a generalized way, it is known that the system won't be perfect (see the No Free Lunch Theorem), so testing human interface factors should allow better optimization of the system. It will also allow stakeholders to see the user reaction to failures and assess their tolerance.

Many systems aim to replace the actions of non-expert users with the actions of an automated system. One goal of acceptance testing that might be required is to assess quality of an AI system in comparison to one or more such users. When this is the goal, it is to remember to consider the human aspects that may affect performance, for instance the time of day, or other environmental factors that humans are susceptible to. It is also important to ensure accuracy is also measured in conjunction with performance. As outlined in the automation bias section, automation can just result in systems making bad decisions faster than humans, and humans assuming these decisions are correct.

2.5 Metrics for Testing AI-based systems

Confusion Matrix

While it is possible to describe the accuracy as the percentage of classifications that were correct, it does not tell the full picture, as the impact of incorrectly classifying the patient as requiring a procedure (a false positive) could be very different to the impact incorrectly classifying them as not requiring the procedure, when they actually did (a false negative). While this is an extreme example because of the very physical impact on the patient, in practice the impact of a false positive error, as compared to a false negative error, is usually different.

A confusion matrix in a chart which describes the accuracy of a system in a classification context. A simple confusion matrix looks like Figure 2 below.

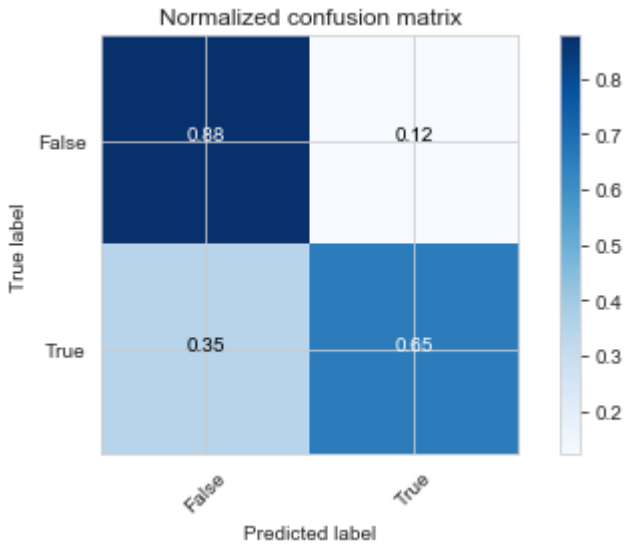

Figure 2: Simple confusion matrix

This matrix is in fact generated from the application of a decision tree algorithm to the Titanic data. On one axis is the predicted labels (true and false, in the simple example above). On the other axis is the true label. The numbers in each box indicate the ratio of results with that true label, which fell into the appropriate predicted label.

Using the matrix above we can see that 88% of people who did not survive were correctly predicted. However, 12% of the people who did not survive were actually predicted to survive, i.e., received false positive errors. Similarly, 65% of people predicted to survive indeed did (true positive) and 35% of people who survived, were not predicted to do so (false negative)

A confusion matrix can be presented differently, using more than one label, using absolute values, and using different colors to highlight different aspects.

Statistical Significance

Due to the nature of probabilistic and non-deterministic systems, the concept of statistical significance can become relevant to software testing, especially in understanding test coverage, for example:

- Expected results can't be determined exactly for a specific test and require multiple executions to establish an average.
- A test analyst is trying to cover different types of users based on their personal characteristics.
- A test analyst needs to ensure all the data features used in a learning model are exercised in testing.
- An A/B test in production is being scoped, and a test analyst needs to determine the number of users to be targeted with new functionality in order to produce the most meaningful results.

Statistical significance is the likelihood that observed test results can be generalized across a wider population of data. There are three factors that affect the statistical confidence that can obtained from a specific number of tests:

- The sample size, or the number of tests. The higher this number, the greater the accuracy of the results. However, this is not linear, as doubling the number of tests does not double the accuracy.
- The variation in responses. If the system responds a certain way 99% of the time, it will require less tests to evaluate, if the system responds a certain way 49% of the time, it will require more tests to determine a statistically significant result.
- The total population size. This might be the total number of permutations of data items in the tests, or perhaps the total number of users. This is usually not relevant unless the test population is greater than a few percentage points of the total population.

A relevant concept is Simpson's paradox. This is a phenomenon in statistics, in which a trend appears in several different groups of data but disappears or reverses when these groups are combined.

The largest number of tests is required when data is stratified or grouped into buckets to ensure it is covered. Any data buckets for stratified sampling typically vary based on data in the input domain that may affect the results, for example:

- A test where the expected results are unknowable should have a statistically significant volume of tests for each possible input value with an important feature.
- A test analyst who is testing facial recognition may make a list of different ethnic backgrounds and aim to cover a statistically significant sample of each.
- A test analyst assessing the quality of a machine learning model may wish to assess whether there is a statistically significant set of training and testing values being used in component testing.
- An A/B test may wish to consider different internet browser or geographies, to ensure they get the most representative results.

It should be noted that the number of such buckets in use increases the number of tests required more than any other factor.

3.0 Using AI to Support Testing

Keywords

Test automation, test maintenance, oracle problem, regression testing, difference testing, Golden Master testing, test data generation, test data synthetization, bug triaging, risk estimation, test generation, test selection, test prioritization, object recognition, identifier selection, cross-browser testing, cross-device testing, visual regression testing

Learning Objectives for Using AI to Support Testing

3.1 AI in Testing

AI-3.1.1 (K2) Summarize why AI is not directly applicable to testing.

AI-3.1.2 (K1) Recall examples of test oracles.

3.2 Applying AI to Testing Tasks and Quality Management

AI-3.2.1 (K2) Give examples of activities and tasks in testing and quality management, that AI can be applied to.

AI-3.2.2 (K2) Give examples of activities and tasks in testing and quality management, that AI cannot be applied to.

AI-3.2.3 (K2) Classify what AI approaches are applicable to test data generation and synthetization.

AI-3.2.4 (K2) Classify what AI approaches are applicable to bug triaging.

AI-3.2.5 (K2) Classify what AI approaches are applicable to risk estimation and fault prediction.

3.3 AI in Component Level Test Automation

AI-3.3.1 (K1) Recall typical problems when generating tests on component level.

AI-3.3.2 (K2) Summarize why test generation on the system level is easier than test generation on the component level.

3.4 AI in Integration Level or System Level Test Automation

AI-3.4.1 (K1) Recall the infinite monkey theorem.

AI-3.4.2 (K2) Explain the difference between monkey testing and fuzz testing.

AI-3.4.3 (K2) Classify what AI approaches are applicable to test generation.

AI-3.4.4 (K2) Classify what AI approaches are applicable to test selection and prioritization.

AI-3.4.5 (K2) Classify what AI approaches are applicable to identifier selection.

AI-3.4.6 (K2) Classify what AI approaches are applicable to visual test automation.

3.5 AI-based Tool Support for Testing

AI-3.5.1 (K2) Give examples metrics that are relevant in a report of an AI-based testing approach.

AI-3.5.2 (K3) Be able to assess AI tool vendor claims for a specific testing task.

3.1 AI in Testing

AI can be used on both sides of testing. It should be tested itself, but it can also be applied to testing and surrounding activities.

The Oracle Problem

The problem in applying AI to testing is the specification of the expected result that the outcome should be compared to. This is called the oracle problem, and it is more complex than it appears to be at first sight.

There are principally two sources for test oracles. The first one is to define the expected result by making the decision as to what the expected result should actually be. This decision is often hard to make (e.g., photo sharing can be SnapChat or Instagram), and is usually the job of the product owner (or even the CEO). The second possibility is to fully understand someone else's decision (i.e., product owner) and to specify the expected outcome accordingly by deriving it from that understanding.

In reality however, things are seldom so clear cut. Often, the general idea is defined by the product owner and remaining gaps are filled by testers and developers, applying common sense. For test automation, however, the expected result needs to be defined completely and unambiguously.

To apply AI to this is currently not possible, as this level of common sense to fill gaps, infer context and interpret intent would need roughly a human-level of intelligence. If an AI would become potent enough for that, it could be used to generate the code (which in itself is a formal specification of the intended behavior), superseding the need for development and testing altogether.

In a broader sense, the oracle problem is the hardest problem of all: telling right from wrong. If this could be achieved in a general sense, all other theoretical problems could be reformulated as instances of the oracle problem and hence be solved. Note that this is not only true for mathematical or logical problems, but also for empirical and philosophical problems, and even for theological problems. For example, write a program that answers the question, "Does God exist?" and ask the test oracle whether the program is correct.

Some of these problems can only be solved in hindsight, some can principally not be solved at all. Obviously, an AI to achieve this feat is not possible yet. Note that the Turing Test described in Chapter 1 is a clever way to circumvent the oracle problem ("What is intelligence?") by applying difference testing.

Test Oracles

Specified test oracles are oracles that are explicitly specified. For manual tests, this is usually the specification or documentation. In typical test automation, this is the widely used assertion. But these assertions usually only apply to the specific test for which they are defined. However, to be usable for test automation or AI (i.e., for new tests), what is usually needed is some sort of formal specification, contract or software model. The problem is the correct level of abstraction of the specification. It must be abstract enough to be practical, but concrete enough to be applicable. After all, the code itself is another form of formal specification. So if the system can be modelled, it often is more efficient to derive an implementation in code than to use the model as test oracle.

Derived test oracles are oracles derived from artefacts that were originally not intended to serve as test oracles, such as alternative implementations or previous versions of the system under test. Especially the latter can serve as test oracles for test automation to detect unintended side-effects

of changes. It is also sometimes referred to as consistency oracle or the testing approach as Golden Master Testing, difference testing, snapshot-based testing, characterization testing or approval testing, that aims to keep the SUT consistent to earlier versions of itself.

Implicit oracles are well-known characteristics that do not need to be modelled explicitly (e.g., the fact that usually crashes are unwanted or software should respond to input in a timely manner).

Testing versus Test Automation

In contrast to what seems intuitive, based on the naming, the goal of testing usually differs from the goal of test automation.

For any given project, the objectives of testing usually include the verification of whether all specified requirements have been fulfilled, validating that the test object is complete and works as the users and other stakeholders expect (see CTFL 1.1.1).

In contrast to that, one of the most widespread use cases for test automation is regression testing or maintenance testing (see CTFL 2.4). Regression testing usually has the goal for a software product (that was initially manually tested) to reduce the risk of unintended side-effects of changes of the code, configuration, supporting systems (e.g., OS) or underlying data.

This allows the usage of the derived consistency oracle for regression test automation, comparing the results and execution characteristics of a test execution to the results and characteristics of the test execution of a previous version of the software. For that to work, it is important to have a deterministic software and execution and only change one thing at a time. For example, one can update the code but not supporting systems, configuration or underlying data. Alternatively, one can update underlying data but not the code. Another important aspect is to be able to ignore irrelevant changes, such as timestamps in the output, or the build number.

3.2. Applying AI to Testing Tasks and Quality Management

Tasks AI Can Be Applied To

As discussed earlier, thanks to the oracle problem, AI cannot be applied directly to testing. However, it can be applied to a lot of the surrounding testing and quality management tasks. In scientific literature and in some testing tools, AI has been applied to:

- Test data generation and synthetization of complex input data
- Bug triaging; e.g., of automated bug reports
- Anomaly detection; e.g., in visual, cross-browser, or cross-device GUI testing
- Risk estimation and fault prediction before new releases
- Effort estimation for manual testing
- Test generation
- Defect analysis
- Test optimization and unification

- Test selection and prioritization in case of time and resource constraints
- Object recognition and identifier selection for creation and maintenance of regression tests
- Identification of visual artefacts for cross-browser, cross-device or visual regression tests

It should be noted that this is a field that overall is still in heavy development, where yet no clear best practices have evolved. The examples given here are either from publications of companies or ongoing scientific research projects.

Tasks AI Cannot Be Applied To

Other activities and tasks in testing and quality management are so challenging, that AI in its current form cannot be applied to it. These include:

- Specification of test oracles (as discussed above)
- Communicating with stakeholders to clarify ambiguities and retrieve missing information
- Discovering contradictions and inconsistencies in the product, specification or documentation
- Suggesting improvements to the user experience and user interface
- Suggesting improvements to underlying processes and procedures
- Challenging assumptions and ask uncomfortable questions
- Understanding user needs

Some of these tasks AI will probably never be applied to, as an AI that is potent enough to be able to be applied to these tasks would be applied elsewhere first, making these tasks obsolete in the first place (see section 3.1).

Using AI for Test Data Generation

Depending on the use case and the concrete system under test, the acquisition of test data might be an issue. If so, AI might be able to help in those situations. The concrete scenario is highly dependent on the actual use case, and more often, a heuristic or (pseudo-) random number generator might be more appropriate to be used for data generation. This approach is called fuzz testing.

There are examples where AI approaches are a better fit for the need. One such scenario is the generation of highly structured input data. For example, a genetic algorithm was used to generate valid JavaScript code (see Holler et al.). This JavaScript was used as test input for JavaScript engines, such as the ones built into Mozilla Firefox and Google Chrome. The oracle was implicit; i.e., the script engines were expected to not crash. Any such crash typically reveals a security issue, and is of highest interest for the developers. The AI was used to generate syntactically valid JavaScript that would pass the initial verification step and reach deeper into the system.

Another scenario would be to use a genetic algorithm to generate valid images or syntactically valid XML (see Havrikov et al.), JSon or any other non-trivial input format, again to check for robustness and security.

To see an example of this in action, visit the website “This Person Does Not Exist” that consists of an AI that, upon opening the website, generates an image of a human face, but a face of a non-existing human. That image is at first glance indistinguishable from a photograph. The image is produced by a

GAN (generative adversarial network), which actually consists of two neural networks. One generates an image; the other one rates how realistic the image is and thus challenges the first neural network to improve on its output.

Using AI for Bug Triaging

Clustering algorithms, such as k-nearest neighbors, can be used in case there are many similar bugs in a bug tracking system with a high volume of bug reports. This is sensible for automated bug reporting systems (e.g., for Microsoft Windows, Firefox or similar) and can also deliver value on large projects with hundreds of software engineers. These clustering algorithms can remove duplicates and with that, point to especially critical bugs that are responsible for a large percentage of the bug reports (see Kim et al.). They can also suggest developers who are best placed to fix defects, based on the defect content.

The way these clustering algorithms work is highly dependent on the available input data (e.g., the diagnostics data associated with the bug report).

Using AI for Risk Prediction and Fault Estimation

Other applications of AI in scientific research have been targeted at analyzing change patterns in the software repository and comparing those changes to the history. By correlating these yet unreleased changes with changes to earlier releases, and combining those with the number of defects found post release, it is possible to both give risk predictions for the release (e.g., number of expected bugs to be found) as well as to point to most problematic changes, allowing for a more targeted manual testing and review process (see Zimmermann et al.).

This approach has an effect similar to the pesticide paradox (see CTFL, section 1.3). The effectiveness of this approach decreases over time, just as pesticides are no longer effective at killing insects after a while.

Also, in pointing to most problematic changes, this approach is somewhat related to the goal of test prioritization, as discussed later.

3.3 AI in Component Level Test Automation

AI in Component Level Test Generation

As was indicated earlier, AI can be used to generate tests. The main limitation is the specification of the test oracle. When generating automated regression tests, results of earlier versions of the system under test can serve as test oracles.

Test generation on the code, unit, module, or component level (see CTFL, section 2.2.1) faces the following challenges:

- Pre-conditions may be complex. Generating tests from code makes it very challenging for AI to understand the state of the software and its data, and any necessary dependencies. For example, the code might expect an established database connection to exist, the specific properties to have been already loaded, etc.

- Parameters may be of complex types. For example, a code function may expect to be passed a complex datatype of sorts, such as a user object or a list of arrays of sets of valid email addresses.
- The oracle problem. Since most of the inner state of the system is available or reachable, the consistency or history oracle will likely show too many non-meaningful and volatile differences (false positives), which reduces the value of the results. Even the implicit test oracle is often defunct, because exceptions and crashes may be non-problematic. This is the case, for example, when robustness is not an issue and the code is being passed invalid data by the test (as no other part of the system does).
- Unclear testing goal. When generating tests for a specific function in contrast to the whole system or a large part of the subsystem, it is unclear when a specific function is tested enough in comparison to the rest of the system. So, test prioritization needs to be incorporated.
- Unclear output optimization. As the test targets code, the expected output of test generation is usually also code. Therefore, many additional hard questions without agreed upon answers arise, such as:
    - What is an optimal test length?
    - How should variables be named?
    - How should code be structured into methods and sub-methods and how should they be named?
    - When is code readable versus spaghetti-code?
- Unclear next action. With a wide variety of methods and functions available and usually most of the system reachable (by breaking the Law of Demeter, see Lieberherr and Holland), it is hard to decide what method or function to call next. Also, there are often implicit protocols in place (i.e., call open before close), for which breaking them makes no sense.

Despite all those challenges, there is some ongoing research in this area, resulting in free-to-use open source tools that generate unit tests for existing code. One such tool is EvoSuite, that can be used on the command line and even comes with plugins for Eclipse, IntelliJ and Maven.

AI in System Level Test Generation

When generating code on the system level, many of the aforementioned challenges simply do not arise (see also Gross et al.):

- When the complete system is started, it usually creates all the conditions needed to be in a correct state; e.g., it establishes database connections, loads configuration, etc.
- On the system level, especially on the GUI, the input parameters consist of only primitive types; i.e., simple point and click user actions and strings. Even when testing technical system interfaces, input formats are usually very well defined, making it much easier to generate valid input data (see section 3.2.3).
- Since we have a clear system-boundary, the consistency oracle can be used to verify the consistency of that system-boundary. Differences on that boundary tend to be meaningful and, in any case, have a much lower noise ratio.

- Implicit test oracles can be used; e.g., typically the system should respond within a certain amount of time and should not crash, no matter what the input is.
- It is easier to define testing goals in relation to the overall system, as the system is in consideration anyways.
- The format of the output can be much less formal; e.g., as an Excel document, listing the individual test steps. Even if outputted as code, the requirements regarding code quality on this level are much lower, and spaghetti code is usually acceptable.
- The available and sensible set of next actions is often obvious. If not, this is possibly also an issue of the user experience design.

3.4 AI in Integration Level or System Level Test Automation

As already detailed in the last section, test generation on the system level is often much easier for a wide number of reasons. One of these reasons is that the set of available next actions is much clearer, another one that these actions are primitive; i.e., they consist mainly of clicks and text input. In fact, both of these make it so easy to generate tests, that this is possible on a purely random manner, without using AI at all.

Monkey Testing Versus Fuzz Testing

The infinite monkey theorem states that a monkey hitting keys at random on a typewriter for an infinite amount of time will eventually type any given text, such as the complete works of William Shakespeare. Assuming actual randomness, the probability for any given sequence of characters to occur is non-zero, meaning that it will eventually happen, given long enough time. So, in eternity, the monkey will not only create Shakespeare's Hamlet, but also infinitely many variations of it.

Both from this theorem and the typically nonsensical way of interacting with the system, the term *monkey testing* was derived. Note that according to the monkey theorem, monkey testing already fully solves the problem of generating any possible test for the system, given a long enough time. This essentially means that adding heuristics and AI to the monkey only makes the approach more efficient; i.e., the monkey arrives at more sensible tests faster.

Monkey testing has been used since the 1980s to test systems for robustness and stability. Note that monkey testing is only sensible in conjunction with the implicit test oracle that a system should not crash, no matter what the user does. If this assumption does not hold, monkey testing does not make sense.

Fuzz testing is the usage of a (pseudo-) random number generator to generate test data. In a sense, the user interaction (clicking, swiping, etc.) can be regarded as a special type of test data. Test user input (e.g., text entered into text fields) is randomly generated test data. In that sense, monkey testing is a special case of fuzz testing.

AI For Test Generation on the System Level

The fact that monkey testing is a special case of fuzz testing also means, that similar AI approaches can be applied. For example, genetic algorithms can be used to generate variations and combinations of existing (recorded) test cases. In addition, neural networks can be used and trained what a typical user would do (i.e., what fields and buttons would be used in what sequence). Note

that the knowledge incorporated in such a neural network is generally not bound to a specific software, as it represents user experience design rules, such as Fitts's law.

One could also describe using a GUI as being like playing a game: In any given GUI state, the AI has to decide what move it wants to do next (i.e., what user action to execute). However, in contrast to a game where the clear goal is to win, in testing, the goal is not so clear. So special consideration has to be given to the definition of the goal. Additional to manually defined soft testing strategies, the goal for the AI has to be measurable in a fully automated way. Therefore, typically goals such as a high overall code coverage or similar are often used.

These approaches can even be combined. An evolutionary algorithm starts off with a seed population. This seed population can either be humanly created (e.g., recordings of user stories), or it can be randomly generated. The better the seed population, the better and faster the results. So, if generating the seed population (partly) randomly, a neural network can be used to generate more human-like tests.

AI For Test Selection and Prioritization

Many projects face a situation where they have simply too many automated tests to run on a regular base. Even with parallelization, the necessary computing power and associated cost is just overwhelming, and on top of that, test results need to be dealt with. Due to their very nature of relying on operating system, platforms, third-party libraries, separate services and network connections, tests on the system level tend to be brittle. Someone has to examine those failures and verify that these are indeed false positives. That additional cost alone justifies test selection and prioritization. Current trends, such as executing tests after every commit, amplify that need.

Test prioritization is related to risk estimation (see section 3.2.6), as the usual goal is to execute tests that have a high probability to reveal unintended side-effects. In contrast to risk estimation, which usually works with static artifacts and their history as preserved in the source code repository, test prioritization can work with runtime information, such as different types of code coverage and test effectiveness (e.g., as revealed by mutation testing). It can further be improved by also considering historical information about the number and criticality of past defects that were revealed by each test (if that information is available, see Jürgens et al.).

Depending on which available data the overall test selection and prioritization system has set up, it can be created using either supervised or unsupervised learning methods. Supervised learning methods would require, for example, the availability of an association of past test performances, such as how many defects a test revealed. Unsupervised learning methods are used if no such labelling is available.

AI For Object Identification and Identifier Selection

Object identification is well-known problem for test automation engineers. Both when simulation user interaction (e.g., when clicking a certain button) and when verifying a certain SUT property (e.g., the text value in an input field), it is necessary to reliably identify different objects or elements on the GUI. This is usually done by an unambiguously identifying piece of information, such as the XPath, the label, and internal id or the X/Y coordinates.

Now, if the chosen identifying piece of information changes either over time (e.g., the button label is changed) or within different environments (e.g., on a screen with a different resolution), the object in question can either not be identified anymore or be identified falsely (e.g., the wrong text field is

used). This typically results in false positives. Depending on when that problem arises, it can even be hard to detect (e.g., when a text is entered in a wrong field early during test execution).

AI can be applied to either help identify the correct object using a multitude of identification criteria (e.g., XPath, label, id, class, X/Y coordinates), using the "looks" of the object by applying some form of image recognition, or by choosing the historically most stable single identification criterion. All of those approaches have been used in the past. However, none of these does address the principal underlying problem: the continual change of the software. So, if the correct object is identified by a multitude of criteria, the approach is robust against changes to one or even multiple of those criteria. However, if such changes occur, the underlying baseline then needs to be updated to reflect those changes. Otherwise, the confidence in the identification will decrease and after multiple such changes occur to different criteria throughout the lifetime of the software, at some point the remaining unchanged criteria will not yield a high enough confidence—thus only postponing the problem, not solving it. The same is true for image recognition: if the past and present image increase in difference, at some point the remaining confidence will not be high enough. And also, for choosing the historically most stable single identification criterion—even if it was the most stable one, at some point it might still change.

AI For Visual Test Automation

Visual test automation has a much narrower goal than general test automation. It aims solely at identifying visual regressions or visual differences on different platforms (cross-platform and cross-device testing). This include unintended changes to the layout of a GUI, the size, position, color, font or other visible attributes of objects, whether some objects are inaccessible or hard to detect, visual artifacts of any sort or any other issues with the visual appearance of the GUI. Visual test automation does not aim to test functional or business aspects of the software.

For visual test automation, image recognition systems and supervised learning algorithms are used and often combined.

3.5 AI-based Tool Support for Testing

Relevant Metrics in an AI-based Testing Approach

Since an AI-based testing approach is an AI system like any other, principally the same metrics can be applied as detailed in section 2.6. Special consideration has to be given to the bias/variance trade-off. As was detailed earlier, one of the common assumptions for AI-based systems to counter the No Free Lunch Theorem is that the past is a good predictor of the future if all relevant variables remain constant. However, this is usually not the case.

For example, with risk prediction and fault estimation, you have to be aware of both the pesticide paradox and the self-fulfilling prophecy, which ironically have opposite effects: Focusing the test effort on specific parts of the system will first have the effect to show more defects and higher risk, because the system is more intensely scrutinized at these parts (self-fulfilling prophecy). After some time, as the quality of the code and the awareness of developers and manual testers increases, the pesticide paradox will kick in, and the effectiveness of the prediction will decrease. Similar effects will show with AI-based test selection. This illustrates the bias/variance trade-off.

For AI-based approaches, one has to constantly monitor the performance indicators, like false positives (type-I errors) and false negatives (type-II errors). A typical problem in this scenario is, that

the rate of type-II errors (missed bugs) often cannot or not directly be determined in a life system (e.g., it is hard to know how many bugs were not detected).

Assess Tool Vendor Claims

According to a report from MMC from 2019 on some 2,830 AI startups in 13 EU countries, when reviewing the activities, focus, and funding of each firm, in 40 percent of the cases they could find no mention of evidence of AI. It can probably be assumed, that a similar rate of wiggle room in what constitutes AI is used when describing software products, such as test tools. Therefore, a healthy amount of skepticism should be applied, when reviewing vendor claims.

As was discussed in section 3.1, it is generally not possible to directly use AI for testing. However, as was also discussed in subsequent sections, there are a number of tasks that AI can be used for. When considering licensing a tool, that the vendor claims is AI-based, a number of considerations have to be made. In general, whether it is really AI or not is an academic question, as long as the tool delivers in the way the vendor promised. However, this is exactly what should be critically scrutinized. Also, management expectations might differ for an AI-based testing tool, and may even be unrealistic, putting tool advocates in challenging situations.

Generally, many tools perform reasonably well for small examples, but scale poorly to larger projects. Especially when using an AI-based tools, many problems get emphasized with scale.

Configuration of the System

Many AI systems require expert knowledge to set all parameters to sensible values. Many of these parameters are specific to the project and the used AI approach. This is why companies like Microsoft, Facebook, Google and the like have whole departments dedicated to tuning and researching in that area.

Some of the configuration parameters of an AI-based system are independent of the AI approach. For example for a test generation system, these parameters include the number of test cases to generate and how they are organized (e.g., the number of suites and modules). How long individual test cases may become (e.g., one test case that covers everything, or many small test cases) affects runtime, effectiveness and readability of test cases. Determining the metric the AI optimizes for directly influences the goal, effectiveness and ultimate output. The time and computation budget of the AI system effects both the generated costs as well as the generated results. Here it is preferable, if the AI system can improve on results from earlier runs.

Other configuration parameters are specific to the AI approach that is being used. For example, the performance of a neural net is highly dependent on the number of layers, perceptrons per layer, and chosen features of the data, whereas the performance of a genetic algorithm depends on the number of individuals per generation, what mutation operators, and what fitness function are being used.

As you can see, many of these questions cannot be answered in a general way but need to be evaluated in the context of a concrete project and a concrete AI approach. This means that you will probably need additional training or consultancy to use the system properly. Does the vendor help with that? How? What is the additional cost to be expected?

Return on Investment (ROI)

While many AI systems work well with small toy examples, one of the major questions to ask is: How much do we have to invest, in order to make the system work well for us, at scale? And in relation to that: What are the benefits of the tool? How much do we gain or save in money eventually? As often effects are subtle or unobtrusive, how can the return be measured? Is it measurable at all? Is it an improvement to an existing process, or does it give an ability, that we didn't have before (i.e., without AI)?

As for the investment to such a new system, there are many questions to ask and considerations to make: Do we need special knowledge (see above)? How much computation power is needed for training or execution of the system? Do we need more or special hardware? How many training examples (data) are needed? How much and which programming skills are needed? How often does it need to run (e.g., before every release or after every commit)? Does its added value decrease over time? How much additional effort is created by the tool itself (e.g., by manually going through false positives), and how much additional risk introduced (e.g., by false negatives)? Some other forms of investment (e.g., changing existing processes, effects on maintenance, etc.) will be detailed below.

Effects on Existing Processes

What effects will this system have on existing processes? For example, do you have to recreate or migrate artifacts, like existing automated tests? Does it integrate well with existing infrastructure, like your existing testing and reporting systems, CI/CD systems, release systems, documentation systems, and/or ticket systems? Will it make people change their behavior (e.g., blindly trusting the system via complacency/automation bias; see subsection 2.2 Automation Bias)? Will people need additional training? Might there be unintended long-term effects?

Sensibility of Test Cases

When generating test cases, it is important to note that different test cases bring different merits. It might be hard for AI to understand how typical users use your specific software. So, questions to ask are: Are the generated tests realistic in terms of typical usage? Are the main business risks covered; i.e., do the generated test cases cover relevant user scenarios? Is the data used (or generated) representative of typical data? Is it even possible for the AI to generate realistic test cases without additional training? How is such a training possible; i.e., what is the input for the AI system in terms of training data? Does it merely suffice to record test cases?

Test Case Explosion

Test case explosion is the combinatorial explosion of the number of possible combinations of different inputs in software testing (a problem that principally has nothing to do with AI by itself). But as AI generates test cases or input data, it becomes important to watch out not only for the number of test cases, but also for the individual value that they add. And that value typically decreases with the number of test cases. For example, the first test case is much more valuable than the thousandth. More test cases come with higher maintenance cost, slower execution, higher complexity in life-cycle management and many more hidden costs. These costs should be outweighed by the benefits the test cases add. This should be closely monitored, also in regard to the maintainability, which we discuss next.

Maintainability

After generating all those test cases, how do you maintain them with reasonable effort in relation to their value? As the pesticide paradox tells, the ability of test cases to find defects decreases over time, while the maintenance effort often increases. Also, generated test cases are usually hard for humans to understand (e.g., determining the goal of any given test case), and therefore hard to maintain. Sometimes a small detail of a test makes it valuable (e.g., in regard to coverage by checking a specific option). When maintaining the test, if the maintainer is oblivious to what makes it valuable, a valuable test can easily be turned into a useless deadweight.

Severity of the Defects Found

When AI generates test cases and detects technical or functional defects, these defects have varying severity; i.e., how likely is it that a customer will face these defects? AI can be very unhuman in its behavior (see also sensibility of test cases) and come up with test cases that only have theoretical value, as usually no human would use the software in such a way. Defects found with such test cases can even have a detrimental effect, in that they divert focus from more customer-related tasks or spam the ticket system. Typically, what is important is not a bug count, but a risk analysis or customer impact analysis. On the other hand, such unusual behavior is often the way adversarial agents try to access a system.

Summary

In summary, it can be said that due to the oracle problem, manual testing cannot be replaced by AI. But AI can improve efficiency and effectiveness of manual testing and many other QA related tasks. However, tool vendor claims should generally (as for any other tool) be critically scrutinized, and an extensive tool evaluation is recommended. When doing this evaluation, make sure to not perform it on a (vendor delivered) toy example, but on a real-world project. As a tester, you should have the expertise to critically test not only your software, but also the vendor tool.

4.0 Appendix

4.1 Glossary

artificial intelligence: Manually designed intelligence (i.e., in contrast to biologically evolved).

A/B testing: A type of testing that is run with two equivalent samples, changing one variable.

actuators: Components of a system which enable it to influence the environment.

adversarial actor: A person with adversarial objectives relating to a system; e.g., a hacker.

agency: A property of a test environment, whether it contains one or more agents.

approval testing: Synonymous to golden master testing, in that changes to the golden master need to be approved.

automation bias: When a human decision maker favors recommendations made by an automated decision-making system over information made without automation, even when the automated decision-making system makes errors.

bias: 1) When a statistical model does not accurately reflect reality; or, 2) When being in favor of or against one thing, person, or group compared with another, usually in a way considered to be unfair. Both can occur independent of one another; e.g., sociological bias that accurately reflects the modelled reality as reflected in the data.

blind test: A testing procedure designed to avoid biased results by ensuring that at the time of the test the subjects do not know if they are subject to a test or are part of a control group. Also **double-blind**, where those evaluating the results are also not aware of whether they are evaluating test results or a control group.

bug triaging: The process of grouping and maintaining (e.g., removing duplicates) defect tickets.

characterization testing: Synonymous to golden master testing, in that the SUT is tested for "characterizing properties," namely the result.

clustering: Identifying individuals to belong to the same group or share some characteristic trait.

confusion matrix: A table that summarizes how successful a classification model's predictions were; that is, the correlation between the label and the model's classification. One axis of a confusion matrix is the label that the model predicted, and the other axis is the actual label.

correlation: A correlation between things is a connection or link between them.

cross-browser testing: Executing the same test with multiple browsers to identify differences

cross-device testing: Executing the same test with multiple devices to identify differences

data feature: A measurable property of the phenomenon being observed, usually an input to a machine learning model.

decision trees: An explicitly modeled way to arrive at a specific resolution, given certain decision parameters.

deep learning: A specific arrangements of a neural network to contain many neural layers.

deterministic system: A deterministic system is a system which, given the same inputs and initial state, will always produce the same output.

difference testing: Testing approach that explicitly checks for changes in relation to a reference instead of, for example, checking for “correctness”.

discreteness: A characteristic of an environment which reflects whether it can be divided into zones or measures, as opposed to an environment which can be continuously measured.

drift: Drift is the concept that the correlation between the inputs and the outputs of an AI system may change over time, including situations where training data labels change over time.

episodicness: A characteristic of an environment which reflects whether it has dependencies on previous states.

expert knowledge-based system: System, where knowledge (usually from subject matter experts) is explicitly represented in the system (e.g., via decision trees or rules).

external validity: Using external data or benchmarks in order to validate the quality of a system.

fuzz testing: A random-based testing approach, where inputs are generated or changed (fuzzed).

golden master testing: A difference testing approach, where the reference is a previous version of the system. The term "Golden Master" refers to the original audio recording used to copy vinyl discs from.

human in the loop: An AI system monitored by a human, or where a human approves each decision.

hyperparameter: A parameter whose value is set to configure the learning algorithm before the learning phase.

identifier selection: The selection of the unique identifying attribute from a machine readable representation (e.g., HTML, CSS) used during user interface automation.

internal validity: Using the input data to a system in order to validate it’s quality.

regression model: A type of model that outputs continuous (typically, floating-point) values.

machine learning: A program or system that builds (trains) a predictive model from input data. machine learning also refers to the field of study concerned with these programs or systems.

machine learning model: A trained model to make predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model.

metamorphic relations: Probabilistic relationships between the inputs and outputs of a system.

metamorphic testing: A testing technique which uses metamorphic relations at a pseudo oracle.

model training: The process of determining the ideal parameters comprising a model.

neural network: A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.

object recognition: In user interface automation, the ability of the automation tool to recognize user interface objects.

observability: The degree to which the AI system can monitor and observe its environment.

oracle problem: The inability to generally identify the correct output of a system.

overfitting: Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.

probabilistic system: A system where the occurrence of events cannot be perfectly predicted.

reinforcement learning: A machine learning approach to maximize an ultimate reward through feedback (rewards and punishments) after a sequence of actions.

risk estimation: The process of identifying system quality risks relating to a testing activity.

sensors: A component used by the system to perceive the environment.

snapshot-based testing: Synonymous to golden master testing, in that the golden master is the snapshot.

staticness: A characteristic of an environment which reflects the degree to which it changes without the AI system taking an action.

statistical significance: The likelihood that observed test results can be extrapolated across a wider population of data.

supervised learning: Training a model from input data and its corresponding labels.

symbolic AI: Formalizing reasoning through mathematical logic and automatic deduction techniques.

test automation: The use of software to perform or support test activities, e.g., test management, test design, test execution and results checking.

test data generation: The automated generation of data for testing purposes.

test generation: The automation production of test procedures

test maintenance: The process of updating an automated test following desirable changes to the system under test which cause the test to fail.

test prioritization: The process of deciding which tests should be run first.

test selection: The process of deciding which tests should be run

training data: The subset of the dataset used to train a model.

type I error: A false positive

type II error: A false negative

underfitting: Producing a model with poor predictive ability because the model hasn't captured the complexity of the training data.

unsupervised learning: Training a model to find patterns in a dataset, typically an unlabeled dataset.

variance: A machine learning model's sensitivity to specific sets of training data

visual regression testing: Test automation which aims solely at identifying visual regressions or visual differences on different platforms

4.2 References

Websites

EvoSuite: http://www.evosuite.org [accessed: June 15, 2019]

Google Machine Learning Glossary: https://developers.google.com/machine-learning/glossary/ [accessed: June 15, 2019]

ISTQB Glossary: http://glossary.istqb.org [accessed: June 15, 2019]

Kaggle (2019). https://www.kaggle.com/c/titanic [accessed: February 1, 2019]

Sci-Kit Learn: https://scikit-learn.org/stable/index.html [accessed: June 30, 2019]

Style-based GANs – Generating and Tuning Realistic Artificial Faces: https://www.lyrn.ai/2018/12/26/a-style-based-generator-architecture-for-generative-adversarial-networks/ [accessed: June 15, 2019]

This Person Does Not Exist: https://thispersondoesnotexist.com/ [accessed: June 15, 2019]

Books and Articles

Barr, E.T.; Harman, M.; McMinn, P.; Shahbaz, M.; Yoo, S. (2015). "The Oracle Problem in Software Testing: A Survey," *IEEE Transactions on Software Engineering*, volume 41, number 5.

Davis, K.; Christodoulou, J.; Seider, S.; Gardner, H. (2011). "The Theory of Multiple Intelligences," in *The Cambridge Handbook of Intelligence*, Cambridge University Press.

Donaldson, A. (2018). "Metamorphic Testing for Non-testable Systems." https://www.testandverification.com/wp-content/uploads/2018/VF_2018/Alastair_Donaldson-GraphicsFuzz_ICL.pdf [accessed: July 21, 2019]

European Commission (2018). "Draft Ethics Guidelines for Trustworthy AI." https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=57112 [accessed: February 1, 2019]

Fitts, P. M. (1954). "The information capacity of the human motor system in controlling the amplitude of movement," in the *Journal of Experimental Psychology,* volume 47. https://psycnet.apa.org/doiLanding?doi=10.1037%2Fh0055392 [accessed: July 1, 2019]

Fortmann-Roe, S. (2012). "Understanding the Bias–Variance Tradeoff." http://scott.fortmann-roe.com/docs/BiasVariance.html [accessed: July 6, 2019]

Gardner, H. (1983). *Multiple Intelligences: The Theory in Practice*, Basic Books.

Gross, F.; Fraser, G.; Zeller, A. (2012), "EXSYST: Search-based GUI testing," in *34th International Conference on Software Engineering (ICSE)*, https://ieeexplore.ieee.org/document/6227232 [accessed: July 6, 2019]

Hastie, T.; Tibshirani, R.; Friedman, J. (2009). *The Elements of Statistical Learning, second edition,* Springer.

Havrikov, N.; Höschele, M.; Galeotti, J. P.; Zeller, A. (2014). "XMLMate: Evolutionary XML Test Generation," in the *Proceedings of 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering.*

Hoffman, D. (1998). "A Taxonomy for Test Oracles," http://www.softwarequalitymethods.com/Papers/OracleTax.pdf [accessed: July 21, 2019]

Holler, C.; Herzig, K.; Zeller A. (2012). "Fuzzing with Code Fragments," in the *Proceedings of the 21st USENIX Conference on Security Symposium*, USENIX Association, https://dl.acm.org/citation.cfm?id=2362793.2362831 [accessed: July 6, 2019]

Jürgens, E.; Hummel, B.; Deissenboeck, F.; Feilkas, M.; Schlögel, C.; Wübbeke, A. (2011). "Regression Test Selection of Manual System Tests in Practice," in the *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR2011)*, http://dx.doi.org/10.1109/CSMR.2011.44 [accessed: July 21, 2019]

Kim, D.; Wang, X.; Kim, S.; Zeller, A.; Cheung, S.C.; Park, S. (2011). "Which Crashes Should I Fix First? Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts," in the *IEEE Transactions on Software Engineering,* volume 37, https://ieeexplore.ieee.org/document/5711013 [accessed: July 1, 2019].

Knauf, R.; A. J. Gonzalez; K. P. Jantke (1999). "Validating rule-based systems: a complete methodology," in the *IEEE SMC'99 Conference Proceedings, 1999 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5.

Lieberherr, K. J.; Holland I. (1989). "Assuring Good Style for Object-Oriented Programs", in *IEEE Software,* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.3681&rep=rep1&type=pdf [accessed: July 6, 2019]

McCorduck, P. (2004). *Machines Who Think, second edition,* CRC Press.

Minsky, M., and S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*, MIT Press.

MMC Ventures, Barclays UK Ventures (2019). "The State of AI: Divergence 2019," https://www.mmcventures.com/wp-content/uploads/2019/02/The-State-of-AI-2019-Divergence.pdf [accessed: July 6, 2019]

Nachmanson, L.; Veanes, M.; Schulte, W.; Tillmann, N.; and Grieskamp, W. (2004). "Optimal Strategies for Testing Nondeterministic Systems," https://www.microsoft.com/en-us/research/wp-content/uploads/2004/01/OptimalStrategiesForTestingNondeterminsticSystemsISSTA2004.pdf [accessed: July 21, 2019].

Russell, Stuart J.; Norvig, P. (2009). *Artificial Intelligence: A Modern Approach, third edition*, Prentice Hall.

Patel, K.; Hierons, R. M. (2018). "A mapping study on testing non-testable systems," *Software Quality Journal*, volume 26, number 4.

Vanderah, T. (2018). *Nolte's Essentials of the Human Brain, second edition*, Elsevier.

Wolpert, D. H.; Macready, W. G. (1997). "No free lunch theorems for optimization," in the *IEEE Transactions on Evolutionary Computation*, volume 1, number 1.

Zhang, H. (2004). “The Optimality of Naïve Bayes,” in the *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)*, AAAI Press.

Zimmermann, T.; Nagappan, N.; Zeller, A. (2008). “Predicting Bugs from History,” in *Software Evolution (69-88)*, Springer.