6 - Risks and Test Strategy for AI Systems

6.1 Risks in testing AI

6.1.1 Risks of Testing AI Systems

Testing AI systems poses some additional risks, in addition to the risks associated with non-AI systems. Some of these risks are

- Test condition related challenges
  - With large number of tests (each dataset being a test), how does one verify
    - correctness of tests
    - completeness of tests
  - Lack of a reliable test oracle to indicate what the correct output should be for and arbitrary input
  - Identification of false positives and false negatives. False positives are still easier to identify because failures are to be investigated. False negatives are difficult to identify because these tests are shown as pass
- Test data related challenges
  - In AI test data is equal to a test case, especially for offline testing.
  - A huge amount of test data is ideally required to test.
  - Data availability can be a problem
  - Data quality can be bad, requiring data cleanup
  - Tagged/annotated data is required for training and testing. Getting this type of data is expensive.
  - Corner cases of ML systems are tough and costly to generate
- Cost related challenges
  - For complex applications, the type of hardware required to train and test the offline model may be expensive
  - Energy costs of training DNN is generally very high
- Skill and tool related challenges
  - Skill requirements for testing AI systems are high. The tester needs to understand how AI systems are built and how they need to be tested
  - Lack of structured information, tools and frameworks for AI systems testing
- Domain understanding and bias related challenges
  - Formal proofs of an algorithm's optimal quality do not guarantee that an application implements or uses the algorithm correctly
  - Need for thorough coverage of input cases, based on domain, are required to avoid biases, incomplete coverage and potential for accidents

6.1.2 Risk of Using Pre-Trained Models

Some organizations have made their pre-trained models available and many AI developers make use of them.
For example, ImageNet models such as Inception, VGG, AlexNet, etc. The models may have their own biases and shortcomings which may need to be discovered by testing. Since these are either unknown or undocumented, systems built using pre-trained models may fail in unanticipated ways or produce results that may be sub-optimal in some situations.

Pre-Trained model issues

- In large scale AI applications, large models are generated offline by executing algorithms on large data, that can be reused in a large variety of scenarios
- In these cases where such pre-trained models are used, there is always the risk that the underlying pattern/truth is not fully objective and fully accurate for the problem at hand
- But such pretrained models offer an advantage in scenarios where the data available for the target problem is scarce and costly
- Testing the target system will carry forward the biases and specifies as presented by the pretrained layer
- Some of the pre-trained models available are – imagenet, inception, alexnet, etc.
- Inherited biases which may not be apparent in absence of information about the data used to train the model
- Unknown or undocumented defects may result in failures of systems built using pre-trained models in unanticipated ways or produce sub-optimal result in some situations
- Susceptibility to the same attacks as the pre-trained model for example adversarial attacks. Known adversarial attacks for original model will have a higher likelihood of success on the new model created by transfer learning and will succeed on a pre-trained model with no changes.
- Misconceptions about the similarities of the task and data between pre-trained model and your problem may produce bad results.
- Data pipeline dissimilarities in the treatment of data may result in poor performance of the pre-trained model.

6.1.3 Risk of Concept Drift (CD)

Working models may degrade over time because of the change in the relationship between input features and output. For example, the impact of commercials and various other types of marketing campaigns results in a change of the behavior of potential customers every few months. This is known as Concept Drift (CD). To find out the occurrence of concept drift, periodic testing of working and integrated models with recent data samples is required. For this purpose, retraining the model and repeating the cycle of various tests and validations is performed in the offline and integrated phases. [JB2] There are various ways to handle concept drift, such as:

- Do nothing
- Retrain the model with recent data
- Periodically update the model by using the latest data on the current model
- Learn the change – An ensemble approach where the current model is left unchanged. A new model takes the output of the current model and learns to correct the predictions
- Detect and choose model – detect the concept drift, if possible, and choose an appropriate model
- AI systems may need to be tested again as time passes by because of concept drift
    - The input/output relationship may change over time, thus invalidating the model and the need for retraining
    - Retraining frequency for the model need to be considered if the new data emerges quickly. Example: AI based security tools as threats keep evolving
- Over time, we need to ask: Is the model behaving correctly, as expected?
- AI features are likely to continuously evolve

Concept Drift and Role in Life Cycle

- Concept Drift is the result of constant change of the underlying truth in data over time.
- AI being quest for training systems based on truth in the training data, if the underlying training data changes, it is a cause of concern of older models are used.
- Hence the need for constant retraining of the system on new or (old+new) data to derive an updated model.
- The frequency with which updates of the model should be made is contingent upon the nature of the domain of the problem – e.g. Weather models will be updated hourly, but retail sales will be updated every quarter.

6.1.4 Challenges of AI Test Environment

The test environments for AI systems can be very complex, owing to possible different use-cases, contexts, and various ways and steps of data preprocessing. From the offline testing point of view, the environment needs are more demanding than from the online testing point of view. There is a need for a large size of data storage, high network bandwidth requirement and also for greater computational power to train/run the model.

6.2 Test Strategy

6.2.1 Test Strategy for Testing AI Applications

A real-world AI based application might employ one or more AI and non-AI components. The test strategy for such a system will include conventional test dimensions as well as new factors specific to AI components and their integration with other system components.

Some of the conventional test strategy considerations are:

- **Level of testing required for the system** - unit testing, integration testing, system testing, acceptance testing and system integration testing as described in ISTQB® FL syllabus. [ISTQB-FL2018]
- **Test techniques** - white-box, black-box and gray-box
- **Functional and non-functional tests** - especially security, performance, robustness and scalability tests
- **Use of test automation**

For testing AI based applications, one needs to take into account the following additional aspects:

- **Offline testing (functional)** - Testing the trained AI model is an additional step that needs to be performed as part of SDLC. The skill requirements for testing in this phase also needs to be considered.
- **Offline testing (non-functional)** - Testing the trained AI model for various non-functional aspects. The model needs to be tested for speed, resource utilization, concurrency and load, scalability before deployment.
- **Black-box testing** - most AI models are exposed as APIs and invoked as black-box generally. A critical component missing in AI systems is the lack of test oracle making black-box testing difficult. Techniques like metamorphic testing are getting popular for ML as it does not require an oracle. Given a successful run of test cases, variation to the test cases are performed and the outputs examined. The need for the output to only satisfy the relationships does not require any oracle. [Wiki5]
- **White-box testing** - In general, white-box testing for ML systems is difficult as, except for some simple models, the inner working of the model is neither clear nor accessible. However, recently, some white box techniques have emerged for ML systems. An example is DeepXplore, an automated white-box testing product. Using neuron coverage, the researchers were able to expose thousands of unique incorrect corner case behaviors. [DX1]
- **Data acquisition and preprocessing** - The available data may not always be in shape to be fed to a model for training or testing. This has been discussed in detail in 3.1 Data preparation and processing.

- **Converting of development environment implementations to production** - for example, converting Jupyter notebook into Python code that runs on the server inside a Docker container on a cloud instance. Once it has been done, the model generated can be saved as a file and used in the application.