April 19, 2019

ARTIFICIAL INTELLIGENCE MODEL TO RUN DYNAMIC REGRESSIONS BASED ON GIT CHECK-IN SUMMARIES TO OPTIMIZE PRODUCT DELIVERY TIME

Vinay Rao

Manjunadh Sridharan

Debaprasad Das

ARTIFICIAL INTELLIGENCE MODEL TO RUN DYNAMIC REGRESSIONS BASED ON GIT CHECK-IN SUMMARIES TO OPTIMIZE PRODUCT DELIVERY TIME

AUTHORS:
Vinay Rao
Manjunadh Sridharan
Debaprasad Das

ABSTRACT

Techniques are described herein for an Artificial Intelligence (AI) based model to drive intelligent and dynamic regressions. The AI model may be based on Git commit messages to reduce end-to-end product delivery time.

DETAILED DESCRIPTION

There exist network management solutions comprising a host of micro-services. Platform-specific components may have sufficient code churn across the board. Although many daily builds only result in code change in a subset of micro-services, an entire suite of tests may be run across all the micro-services to ascertain base quality. Moreover, the tests associated with a module that is not affected by a code change are usually successful and are therefore redundant. Accordingly, a solution is described herein to detect the modules in which the code change has occurred and run only the appropriate test tags instead of the entire gamut of tests. This can reduce the amount of time required to certify a given build, for example, from 15 hours to 3 or 4 hours depending on the magnitude of the change.

The solution may be end-to-end (i.e., the solution may be made operational from the time at which the code is checked into the main branch to the time where the entire deployment unit is qualified for its usage). For example, after a piece of code has been checked in, the test tags associated with that module are immediately and automatically determined, and the tests executed. The amount of time required to run all the regression jobs against a build at a functional test level may be reduced by learning the Git-checkin-to-test-tag mapping based on historical data provided to the Artificial Intelligence (AI) model. Because this solution may run in a micro-service environment, the service adjacencies may be identified/learned. Those test modules which are not directly associated

with the code change, but might nonetheless be affected based on historical data associated with this change, may be executed.

This solution provides the ability to predict direct and indirect component adjacencies to run targeted and intelligent regressions without an explicit need for all the associated keywords to be present in the Git commit message. This also enables avoiding any static definitions of dependent services and constantly monitoring the same. Further, this allows for "weighted testing" of end-to-end functionalities in a distributed/multi-service environment by executing relevant module tests before executing other test scenarios. Lastly, because this utility employs "active learning," it is constantly improving its predictive capabilities. A "test tag recommendation engine" may recommend the most probable tags that are associated with the Git commit.

The pre-processing logic extracts the Jira Identifier (ID) from a Git commit to add that detail while adding the data and label for that row of information. The pre-processing logic on the Jira-related information weeds out various irrelevant columns of information. Once the pre-processed data is split into input data (X) and associated labels (Y), it may be passed through the training phase, where certain parameters of the model are tuned to increase its accuracy. Once training is complete, that model may be used for testing.

Figure 1 below illustrates an example training phase.

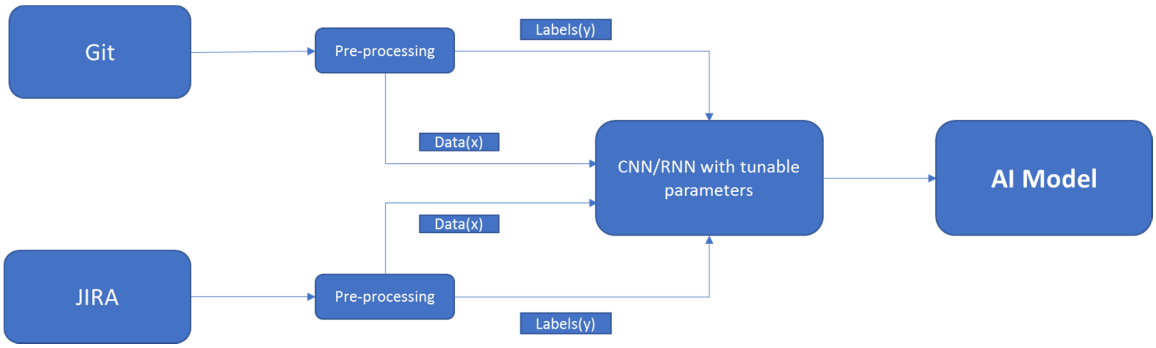

Figure 1

Once the model was built (i.e., it has learnt which label(s) to associate with given text-based input), the model was tested with pieces of Git commit messages that were not provided to the model during training. Its predictions about the associated labels were fairly acceptable. With 105 total labels to choose from and about 2400 total input files (Jira bugs

and Git commits), the validation accuracy was approximately 60% on average. The highest validation accuracy after tuning was 67%.

Figure 2 below illustrates an example testing phase.

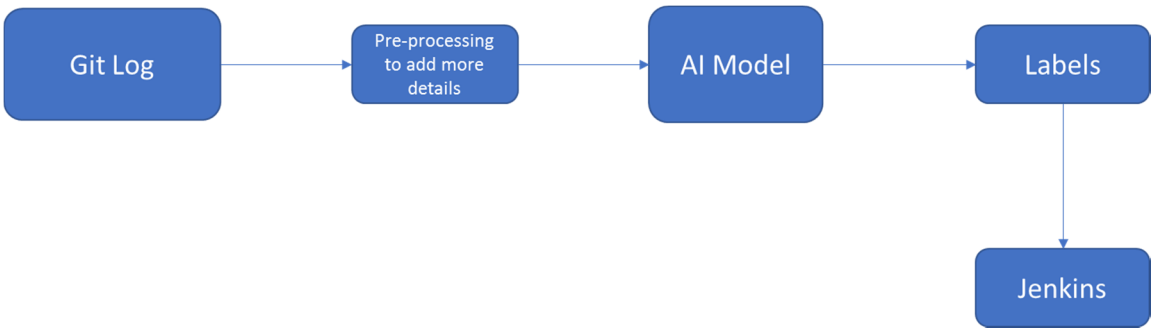

Figure 2

There are two modes of operation. In the first mode ("fully-automated-mode"), the responsibility of executing relevant tests is left completely to the AI tool. This mode may be switched when the accuracies improve beyond the user's definition of acceptable validation accuracy.

In the second mode, the concept of "human-in-the-loop" is also incorporated into the system, whereby the framework provides a human with an option to annotate tags in case of poor matches. This information may feed back into the system in order to provide additional data points during the next training checkpoint.

The most relevant label tends to consistently appear in the top twenty results. Further, the model seems to incorporate adjacent tags in the top twenty results, allowing for selection and running of test tags for modules which are not directly associated with the code change. This information may also help increase test coverage.

The data used in the training phase and testing phase may be completely independent of each other. The raw Git log text obtained from the list of Git check-ins made on a given day may be passed as a list of strings into the model. The outcome for each of the items in the list is a set of the top 3/5 labels.

In summary, techniques are described herein for an AI based model to drive intelligent and dynamic regressions. The AI model may be based on Git commit messages to reduce end-to-end product delivery time.