Updating Deep Specification Mining Model with new traces

Introduction

After building a automata-based model using Deep Specification Miner (DSM), it is possible to update the model with new execution traces after it has been deployed into real-world systems. As we incrementally update the model with new data only once they become available, we are able to integrate new information into our model while avoiding the large cost of retraining the model with the full dataset. The low cost of updating a model with new execution traces entails that newly obtained data can easily be included in an already deployed model.

Incremental Update

When building the DSM model, a feature vector is generated for each node in the Prefix Tree Acceptor (PTA). These features include features based on which method calls preceded the current state, and the probability of the next method call based on a Recurrent Neural Network (RNN) model. Through clustering, these feature vectors are used to determine which nodes are similar enough to be merged. The RNN and information about each cluster is retained to allow updates to the model.

To update the model with new example traces, a feature vector for each node based on the preceding method calls and the RNN is created. These nodes are merged into the existing node clusters of the DSM model based on the feature vectors. To determine if a node can be merged into a cluster, the distance from the node to the representative node of each cluster is computed. If this computed distance is less than the distance from the representative node to the furthest node in the cluster, then the node can be merged into the cluster. If there are multiple clusters that can be merged into, the cluster with the shortest distance is selected.

Evaluation of performance

We use the original input traces used in DSM. Models were built for 11 Java library classes. These library classes have been previously used for the evaluation of specification mining algorithms, and the classes span a variety of categories, ranging from Data Structures, Data Processing (such as NumberFormatStringTokenizer, refered to as NFST), to Message Exchange.

Using the Unix time utility, we compare the real time elapsed of doing an incremental update versus a complete retraining of the model under 2 scenarios:

A. 100 new instances of execution trace is available.

B. a large number of new data (10% of the original dataset) is available.

Class	Time for incremental update	Time for retraining
ArrayList	0 mins 16.169s	1 mins 14.148s
HashMap	0 mins 11.465s	1 mins 58.541s
HashSet	0 mins 10.131s	6 mins 31.687s
HashTable	0 mins 8.425s	2 mins 40.528s
LinkedList	0 mins 10.198s	0 mins 46.559s
NFST	0 mins 9.282s	2 mins 58.362s
Signature	0 mins 4.799s	6 mins 6.858s
Socket	0 mins 24.064s	4 mins 11.705s
StackAr	0 mins 11.336s	3 mins 45.533s
StringTokenizer	0 mins 5.374s	8 mins 6.383s
ZipOutputStream	0 mins 4.611s	1 mins 39.613s

Table 1: Time required for each class when given 100 new instances (Scenario A)

Class	Time for incremental update	Time for retraining
ArrayList	0 mins 22.332s	1 mins 20.716s
HashMap	0 mins 13.769s	2 mins 34.627s
HashSet	0 mins 12.626s	7 mins 35.934s
HashTable	0 mins 8.814s	2 mins 6.845s
LinkedList	0 mins 8.560s	0 mins 33.731s
NFST	0 mins 6.722s	0 mins 34.530s
Signature	0 mins 3.970s	4 mins 54.392s
Socket	0 mins 30.395s	3 mins 51.766s
StackAr	0 mins 10.188s	3 mins 26.245s
StringTokenizer	0 mins 3.896s	8 mins 12.895s
ZipOutputStream	0 mins 5.040s	1 mins 57.773s

Table 2: Time required for each class when given 10% of the original training dataset (Scenario B)

In Table 1, under Scenario A, the cost of an incremental update is low for every model, and is significantly less than the cost of retraining the dataset. In the worst case in our experiments, the cost of retraining a model is over 20 times the cost of incrementally updating the mode.

Under Scenario B, when we only update the model after receiving a large amount of execution traces, Table 2 shows that cost of retraining is still significantly higher than an incremental update.

From the experimental results, we determine that the time to incrementally update the model is cheap. Although we did not explicitly measure it, apart from the time required to retrain the model, the cost of CPU and RAM usage is much higher when retraining the entire model. Our results demonstrate the low cost of incremental updates as well as its feasibility to be used in a real-world setting as compared to retraining the entire model