Additions to Xplacer Workflow

Brendan Badia
Department of Electrical and
Computer Engineering
Northwestern University
Evanston, Illinois 60201
Email: brendanbadia2021@u.northwestern.edu

***Abstract*—For my work here this Summer I was tasked with updating the workflow of the Xplacer work. I detail the components I worked on here, how they function and the improvements they have been able to make to the offline training section of the work.**

I. Introduction

Xplacer is an advice controller that guides CUDA code decisions, introduced in [1]. It does so in two steps. The first is to build a set of features on runtime applications and use it to train an ML model to learn the optimal memory advises (either the use of discrete memory or unified memory with additional memory hints). This is called the offline learning stage. In the second, Xplacer the collects runtime metrics from the target applications and converts them to the feature space used for training. It makes predictions for the CUDA advice to pass during this time. This is called the offline inference stage.

For the Summer of 2020, I was hired to enhance the workflow of Xplacer, specifically the offline training section. There were three main components that I focused on. The first was to convert how models were constructed. Previously, Weka was utilized to build the models. Weka has its own proprietary file formats and is build in Java, which hinders the ability to share models and data. I worked to build scripts that allowed models to be build in python using the Sklearn library. Secondly, I worked on choosing a fileype for models and finding an avenue to publish and share them. I chose the ONNX file format and built models and saved data that can be shared with other researchers. Finally, I was tasked with working on automating the learning process more. One of my scripts allows commands to be passed in to either build a specific model type, optimize over one model type, or attempt to choose the best from a range of models.

I also performed some additional experiments to attempt to utilize cost-sensitive learning to this framework: more details can be found in the other document. Lastly, I wrote some c++ scripts to automate the Weka training in case this was ever desired.

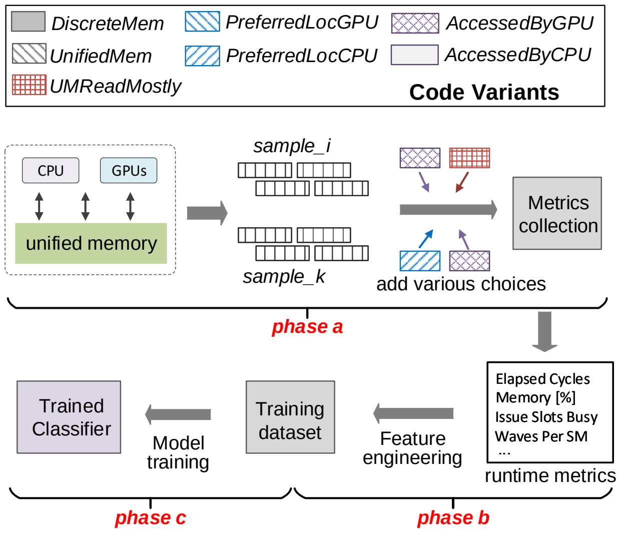


Fig. 1. The workflow of the offline learning in `XPlacer`.

II. Automation for Weka

The previous work utilized Weka to build models in the offline stage [2]. As discussed in the Introduction, it was viewed that moving away from Weka would be beneficial. However, for a paper deadline in August it was decided that automating the Weka portion would be preferable to using the new models generated in Sklearn.

Weka has a command line feature but I found there were some issues with using it to automate all parts of the training process: building models, testing them, and using them to make new predictions. I therefore wrote a script called "WekaInterface" which allowed the automation of these three steps. By passing in the flag "-b", one can build a model (and add additional filters like normalization formatted as described by the Weka documentation). By using the flag "t", once can test a saved Weka model on data and print out relevant accuracy metrics. Finally, by using the flag "-p", one can makes predictions and have them saved in a csv file.

III. Migration to Sklean

The bulk of the work focused on switching to the offline training portion to python scripts using the Sklearn library [3].

A. *Open Neural Network Exchange (ONNX) file format*

One goal that motivated the swtich to Sklearn was to have the ability to more easily share trained models to give other researchers easy access to them. Standardization has long eluded Machine Learning researchers. ONNX was introduced in 2017 to attempt to alleviate this, and is growing in popularity. ONNX files represent different machine learning models that are saved as acylic graphs. Support for ONNX files is offered in almost all major ML libraries, including Sklearn. All files I build with the python scripts save the trained models in the .onnx file format.

B. *GUI*

The first script I wrote was a GUI that allows users to read in data saved as a csv and train various models. The choice of models include SVM, Random Forest, Adaboost, Bagging, Gradient Boosting and Decision trees. The accuracy of the model is tested on the whole dataset as well as the $k-$fold cross validated accuracy, where k is set by the user. These statistics and the time taken to produce the model are printed in the output log. The model generated can be saved in the onxx file format with a name set by the user. Finally, a file can be read in a predictions can be made on it, which can be saved as a user specified csv file.

In the previous XPlacer work, the five ML models tested were the decision tree, random forest, random tree, bagging and REPTree. I compared the accuracy (both on the whole dataset and with 10-fold cross validation) of these Weka models on the largest dataset used for the 2020 paper submission. I also compared the time taken to build the models. In Tables 1,2 and 3 the results of these experiments are shown (note that Sklearn does not have the random tree nor the REPTree model). These were all run using the default hyperparameters for both settings. As can be seen, using both accuracy metrics the sklearn models outperform the Weka models. Furthermore, the time taken to build the sklearn models is significantly shorter than the Weka models. These experiments seemed to suggest the move to sklearn was the right one.

TABLE I
MERGED DATA ACCURACY ON ALL DATA (5376 DATA POINTS)

Model	**Accuracy (Weka)**	**Accuracy (sklearn)**
Decision Tree	91.87%	95.07%
Random Forest	91.22%	95.07%
Random Tree	90.86%	N/A
Bagging	91.06%	94.95%
REPTree	89.28%	N/A

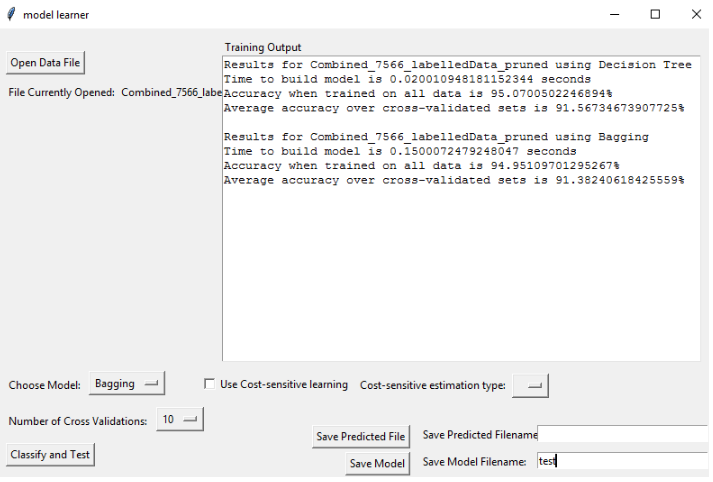
TABLE II
MERGED DATA 10-FOLD ACCURACY (5376 DATA POINTS)

Model	**Accuracy (Weka)**	**Accuracy (sklearn)**
Decision Tree	89.18%	91.34%
Random Forest	91.29%	91.09%
Random Tree	90.79%	N/A
Bagging	91.07%	91.38%
REPTree	89.31%	N/A

TABLE III
MERGED DATA MODEL TRAINING TIME (5376 DATA POINTS)

Model	**Time taken Weka (s)**	**Time taken Sklearn (s)**
Decision Tree	0.18 to 0.19	0.024 to 0.029
Random Forest	2.2 to 2.9	0.68 to 0.72
Random Tree	0.03 to 0.06	N/A
Bagging	0.58 to 0.6	0.148 to 0.155
REPTree	0.06 to 0.09	N/A

1) Cost-sensitive Learning: The GUI also enables users to choose whether or not they want to use cost sensitive learning, and which estimation metric to use. The details for cost sensitive learning (for example, what it is and how each estimation metric works) are described in the other document.


model learner
Open Data File
File Currently Opened: Combined_7566_labe
Training Output
Choose Model: Bagging
Use Cost-sensitive learning
Cost-sensitive estimation type:
Number of Cross Validations: 10
Classify and Test
Save Predicted File
Save Predicted Filename
Save Model
Save Model Filename: test

```
Results for Combined_7566_labelledData_pruned using Decision Tree
Time to build model is 0.020010948181152344 seconds
Accuracy when trained on all data is 95.0700502246894%
Average accuracy over cross-validated sets is 91.56734673907725%

Results for Combined_7566_labelledData_pruned using Bagging
Time to build model is 0.1500072479248047 seconds
Accuracy when trained on all data is 94.95109701295267%
Average accuracy over cross-validated sets is 91.38240618425559%
```

Fig. 2. Example of using the GUI to train various models and save the chosen trained model.

C. *Command Line learner*

The GUI allows users to construct models but does not aid in automating the process. Therefore, I added a python script that is run from the command line which would allow this. The script has three flags which give users the ability to build models in three different ways. For all these flags the model chosen (or the chosen best model) is saved in the ONNX format with a name specified in the command line. The models that are usable are the same as the GUI.

The first flag is for “manual mode” and is chosen by using the flag “-m”. In this case, a specific model type is passed in to the command line and that model is built. The default hyperparameters are used unless the user passes in an additional argument of a list of hyperparameters and values for each.

The second flag is for “optimize mode” and is chosen by using the flag “-o”. In this mode, a single model type is chosen but various hyperparameters are tested to choose the best performing option (in terms of k fold cross validated accuracy) . There are two ways to do this, which are chosen with an additional argument. If option “0” is chosen, a grid search is performed meaning every option is tested and the best performing one is chosen. If option “1” is picked, a random search is performed. In this case, points are randomly sampled from the grid for a specified number of iterations and the best performing point is chosen. The grid (which represents a range of values for various hyperparameters) can be passed in by the user or default values can be used. The number of kfolds used to determined the best performing option is set by default to be 10 but can also be set by an additional argument in the command line.

The last flag is for "automate mode" and is chosen by the flag "-a". In this mode, every single model type is tried and the best performing one (again using k-fold cross validation) is chosen). Again there are additional arguments that need to be passed to specify how this is done. If option "0" is chosen, the default hyperparameters are used for all models. If option "1" is chosen, grid search is performed on all model types. Lastly, if option "2" is picked, random search is performed on all model types. For options 1 and 2, the grid or random search is used to find the best hyperparameters for each model type: the overall best model is the one which performs the best when compared to the others.

D. Sharing Models and Data

By using models stored in the ONNX file format, we are able to more easily share data with other researched. One site to do this is the Data and Learning for Science (https://www.dlhub.org/). I have organized some data and models on github that would be suitable to share on this website

IV. Other additions to workflow

To enable cost-sensitive learning, I added multiple python scripts to label cost data and organize files to allow my to perform my experiments.

References

[1] H. Xu, M. Emani, P.-H. Lin, L. Hu, and C. Liao, "Machine learning guided optimal use of gpu unified memory," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*. IEEE, 2019, pp. 64–70.

[2] F. Eibe, M. A. Hall, and I. H. Witten, "The weka workbench. online appendix for data mining: practical machine learning tools and techniques," in *Morgan Kaufmann*, 2016.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.