White-Box Program Tuning – Supplementary Material

1. Overfitting

Since machine learning algorithms normally produce models as their output, the tuning task of these parameters is usually guided by the execution results of the models (e.g., lower classification errors). Unfortunately, it may lead to *overfitting*, meaning that the tuned parameters produce optimal results on the training data but poor results on the testing data. Note that other programs (e.g., `Canny`) do not have this problem as they are tuning for the final output data but not models tested by different data.

WBTuner provides intrinsic support to address overfitting by combining its execution model with *k-fold cross-validation* [25], a widely used technique for preventing overfitting. Specifically, to tune the parameters in a machine learning algorithm, the user only indicates the k value in the *wbt_sampling*() primitive and provides a validation callback. By doing so, WBTuner will then transparently include k-fold cross-validation during tuning.

The tuning-validation model is shown in Fig. 1. First, the input data is transparently divided to k datasets for each sample run. Moreover, for the xth original sample run, WBTuner spawns $k-1$ more processes, that form a *sampling and validation group* (SVG). If the user intends to collect n samples originally, WBTuner internally creates n SVGs, that is, $n * k$ processes. All the k processes in a SVG share the same sample values for the tuning variables but use different datasets for training and validation to prevent overfitting. As illustrated in the figure, the ith process in the SVG uses the ith dataset for validation and the remaining $k-1$ datasets for training. At the end of the execution of an SVG process, WBTuner invokes the user-supplied validation callback to apply the produced model on its validation dataset and computes the validation error. The validation errors from all SVG processes are then aggregated to drive the remaining steps of the tuning procedure.

2. Benchmarks

In WBTuner, we tune 12 benchmarks and one complex drone control software. These benchmarks are different and widely used. Following are the descriptions. `Canny` [6] is a widely used edge detection algorithm in image processing; `Watershed` [3] is an algorithm for image segmentation; `k-means` [17] and `DBScan` [9] are for clustering data; `Face Rec` is a CSU face identification and evaluation system [5]; `Speech Rec` is from CMU Sphinx for speech recognition [14]; `Phylip` [22] generates phylogenetic trees in bioinformatics; `FASTA` [20] identifies statistically significant similarity between DNA or protein sequences; `TOPN Rec` [19] is a popular data mining item recommendation program; `METIS` [13] performs graph partition; `C4.5` [23] is a decision tree and rule sets based classifier; `SVM` [8] is a supervised learning model for multi-class data classification. `Ardupilot` [2] is an advanced open-source vehicle controlling software which has been installed in over one million vehicles world-wide.

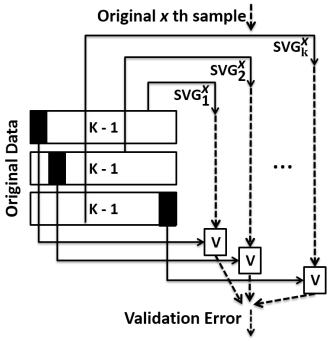

Figure 1: Tuning + Validation Execution Model

2.1 Selected Tuning Case Studies

Due to page limit of technical paper, in this section, we show the details of the rest of selected representative programs under the single core environment and two tuning missions results of Ardupilot in the multi-core environment.

2.1.1 Data Mining

K-means. K-means [17] partitions input dataset into `K` clusters, each holding similar data. The algorithm is shown in Fig. 2. Given the input `objs`, it first generates `K` initial centroids (for clusters) through a non-deterministic random procedure (line 14). It then calls `kmeansCore()` (line 17) to iteratively refine the centroids and the corresponding clusters. The clustering result is stored in `objsAssign`. A data object is put in the cluster of the closest centroid. Although there is only one tunable parameter, `K`, in `K-means`, the clustering result is nondeterministic even `K` is fixed because it also depends on the randomly selected initial centroids.

The tuning code is highlighted in Fig. 2. The quality of clustering result is calculated by the Silhouette score [24]. Higher score means better clustering result. We use MCMC sampling (line 11) to demonstrate using a different sampling strategy. The probability of accept-

ing the current sample is determined by its improvement over the old score. If a sample is accepted, the next sample will be in its neighborhood. Lines 2-5 denote the scoring function for one sample. Observe that it is used for both sampling and aggregation strategies. Note that the scoring function implementation is trivial because Silhouette score calculation is provided by the benchmark. Furthermore, the rest of MCMC sampling and MAX aggregation is transparent. In addition, line 15 invokes a callback to prune the centroids with poor quality based on the algorithm in [18], which prevents the expensive core algorithm execution. Totally there are 5 lines of primitives and 98 lines in callbacks.

```
1 /* User provided callback */
2 double scoring(int sampleIdx) {
3   result = wbt_loadS(objsAssign, sampleIdx)
4   return silouette(result);
5 }
6 /* Source program */
7 void kmeans() {
8   // Initializations
9   objs = input(file);
10
11  wbt_sampling(numberOfSamples, mcmc, scoring)
12  K = wbt_sample(uniform(2, 25))
13  initializeCentroids(centroids);
14  wbt_expose(centroids)
15  wbt_check(checkCentroids)
16
17  kmeansCore(objs, objsAssign, centroids, K);
18  wbt_aggregate(objsAssign, MAX, scoring)
19  visualize(objsAssign);
20 }
```

Figure 2: White-box tuning for `K-means`

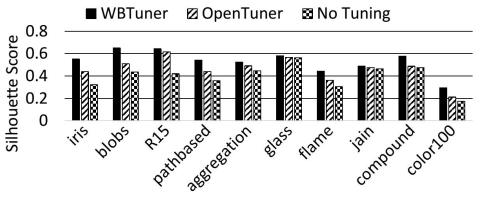

Figure 3: `K-means` tuning scores for 10 datasets

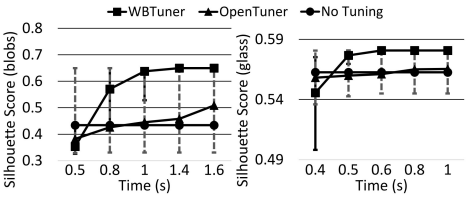

Figure 4: `K-means` tuning score variation

Fig. 3 shows the clustering results when the tuning time is the convergence time of WBTuner. The ten data sets are from [7, 10–12, 16, 21, 26, 27]. Observe that WBTuner consistently produces the best results. The result improvement with OpenTuner over no tuning is 7% on average. The improvement with WBTuner is 38%. In other words, WBTuner is almost six times more effective in tuning the results with the same time. Fig. 4 shows the score variations for the `blobs` and `glass` datasets, in which WBTuner has the maximum and minimum improvement over OpenTuner. For `blobs`, 1 second WBTuner tuning yields much better results for 1.6 seconds OpenTuner tuning. Fig. 5 visualizes the different clustering results. Observe that the WBTuner result is more reasonable. For `glass`, although the two yield very similar final results, WBTuner reaches the result within 0.6 second whereas OpenTuner cannot reach the same score in 1 second.

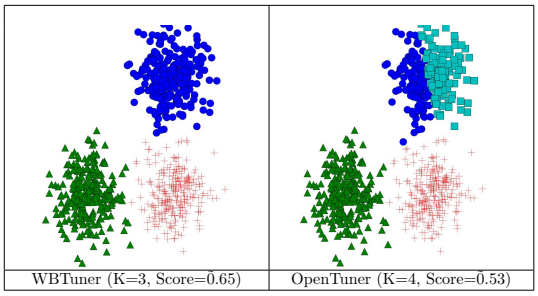

Figure 5: Blobs clustering

C4.5. C4.5[23] is a data mining program. It takes the training data with features and their class labels, and then generates rule sets and a decision tree. The decision tree can be used to classify unknown data. C4.5 has two tunable parameters. Different parameter settings may result in substantially different classification results. Like most data mining and machine learning programs, C4.5 suffers from the overfitting problem (Section 1). Hence, we tune C4.5 with k-fold cross validation supported by WBTuner (with k=10, a typical setting for cross-validation [25]) and compare the result with OpenTuner. Note that OpenTuner does not handle overfitting by default, we extended its implementation to provide the cross-validation as well (using the same k).

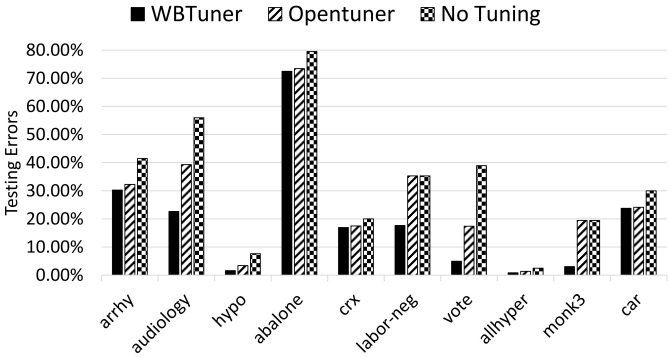

Figure 6: C4.5 tuning scores of 10 datasets

In our experiment, we used 10 datasets from [4]. For each dataset, we divide it to two equal parts, and used the first half to perform training and tuning. The second half was used for testing. Observe the tuning result is a classifier which is used on the testing data to measure its quality. Fig. 6 shows the testing errors for WBTuner and OpenTuner, and the testing error for no-tuning. Observe that WBTuner consistently produces the best results. Fig. 7 shows the score variation for the best and worst datasets. Observe that for audiology, it takes 0.25

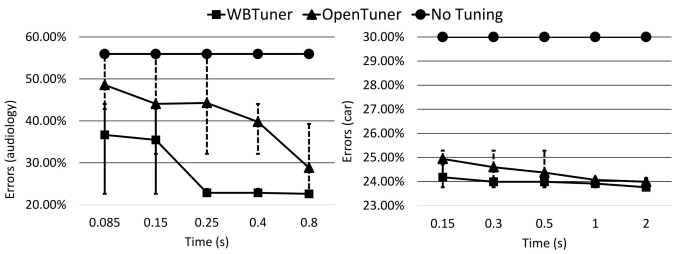

Figure 7: C4.5 tuning score variation

seconds for WBTuner to reach the lowest error while it takes 0.8 seconds for OpenTuner to reach it. The results strongly suggest that overfitting is a prominent challenge in tuning and WBTuner substantially mitigates the problem transparently.

2.1.2 Tuning Drone's Behavior.

Here we discuss the two tuning missions used to tune the parameters of Ardupilot to make it learn the flying behavior of PX4.

Fig. 8 shows both the motors speed and visual results for the first tuning mission. Note that motor speeds represent the key states of a drone. The gray point in the visual results indicates the front of the drone. As illustrated, PX4 first accelerates the drone to a high speed (with the initial spikes of motor speeds close to time 0). It then maintains a stable high speed till until time-stamp 7 (sec). At this time, it decelerates as it reaches the targeted height. In contrast, the default Ardupilot rises very slowly and exhibits tilts and turns (due to some calibrations when taking off). After tuning, Ardupilot is more stable at take-off (i.e., the tilts and turns are avoided). If one looks into the motor speed chart, the spike and the dip (at 7th sec) appear, resembling the PX4's chart. While WBTuner is not able to achieve the same sharpness of the spike/dip as in PX4 due to other un-tuned parameters, the result is promising.

Fig. 9 shows the results of the second tuning mission (The three way points are indicated by A, B and C). When the default Ardupilot reaches the middle way point (i.e., B), it first tilts, turns at the same spot until its head points to the next way point C, and then flies towards C. Intuitively, changing the orientation at point B requires the drone to decrease its speed, and hence leads to a longer mission. Conversely, both PX4 and the tuned Ardupilot avoid turning as much as possible at point B, and rather change their orientation while flying towards C (as indicated by the white curved arrow), consequently finishing the mission in a much shorter time.

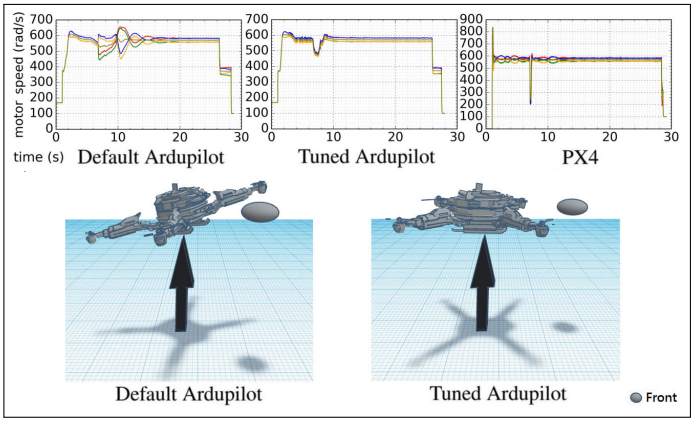

Figure 8: Tuning mission 1

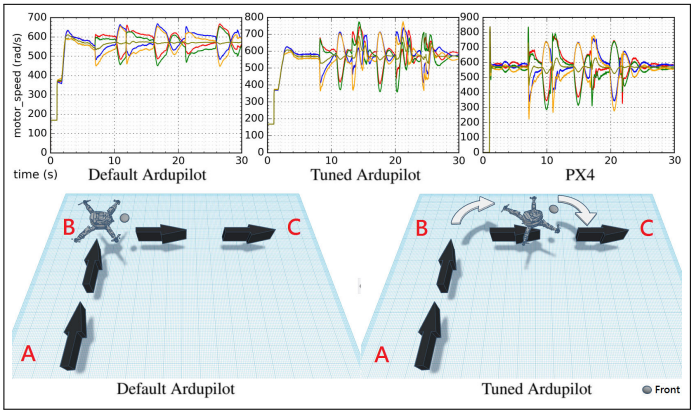

Figure 9: Tuning mission 2

References

[1] An4 database. http://www.speech.cs.cmu.edu/databases/an4/.

[2] Ardupilot. http://ardupilot.org/.

[3] Leptonica image processing and analysis library. http://www.leptonica.com/.

[4] A. Asuncion and D. Newman. Uci machine learning repository, 2007.

[5] D. S. Bolme, J. R. Beveridge, M. Teixeira, and B. A. Draper. The csu face identification evaluation system: its purpose, features, and structure. In *Computer Vision Systems*. 2003.

[6] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6).

[7] H. Changa and D.-Y. Yeung. Robust path-based spectral clustering. *Pattern Recognition*, 41(1), 2008.

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), 1995.

[9] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD, 1996*.

[10] L. Fu and E. Medico. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC*

Bioinformatics, 8(1), 2007.

[11] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *ACM Trnsactions on Knowledge Discovery from Data*, 1(1), 2007.

[12] A. Jain and M. Law. Data Clustering: A User's Dilemma. 2005.

[13] G. Karypis and V. Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[14] P. Lamere, P. Kwok, E. Gouvea, B. Raj, R. Singh, W. Walker, M. Warmuth, and P. Wolf. The cmu sphinx-4 speech recognition system. In *ICASSP, 2003*.

[15] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The alpbench benchmark suite for complex multimedia applications. In *IISWC, 2016*.

[16] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[17] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, 1967.

[18] K. A. A. Nazeer and M. P. Sebastian. Improving the accuracy and efficiency of the k-means clustering algorithm. In *WCE, 2009*.

[19] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *ICDM, 2011*.

[20] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8), 1988.

[21] 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*, 12, 2011.

[22] D. Plotree and D. Plotgram. Phylip-phylogeny inference package (version 3.2). *cladistics*, 5, 1989.

[23] J. R. Quinlan. *C4. 5: programs for machine learning*. 1993.

[24] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 1987.

[25] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1974.

[26] C. Veenman, M. Reinders, and E. Backer. A maximum variance cluster algorithm. *Pattern Analysis and Machine Intelligence*, 24(9), 2002.

[27] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput.*, 1971.