Estimating User-Produced-Multimedia Classification Performance Efficiently with Python

Peter Birsinger
UC Berkeley
peterbir@eecs.berkeley.edu

Benjamin Elizalde
UC Berkeley
benmael@icsi

Karl Ni
UC Berkeley
ni4@llnl.gov

Gerald Friedland
UC Berkeley
fractor@icsi

Armando Fox
UC Berkeley
fox@cs.berkeley.edu

ABSTRACT

When developing a multimedia classification system in preparation for an evaluation such as TRECVID MED or MediaEval, one typically must estimate the error on the evaluation data set from the performance on the development data set, a task made increasingly important with user-produced videos that often lack structure and have high acoustic variability. Statistical bootsrapping reveals the uncertainty of the estimate of a classifier's performance, yet with today's large multimedia datasets, bootstrapping applications, particularly when written in common productivity languages such as Python or Matlab, can become essentially intractable, as we later show.

To combat this, we propose here a methodology to efficiently estimate classifier performance that relies on the ability to generate distributed bootstrapping applications from serial Python. Utilizing an already made Domain-Specific Embedded Language (DSEL) and corresponding DSEL compiler [3] made with the SEJITS (Selective Embedded Just-In-Time Specialization) approach [10], we describe a bootstrapping application in Python to evaluate a multimedia classifier's performance and automatically generate distributed code scalable to demanding data sizes and computational workloads. With this methodology we obtain reasonable bounds for classifier performance predictions on user-produced videos from the 2013 TRECVID MED corpus, showing that reasonbly estimating performance on such data is possible. We then demonstrate the newfound accessibility to such computations for non-performance programmers by exposing the orders of magnitude speedup obtained over the native Python code.

Keywords

Multimedia classification; bootstrapping; SEJITS

1. INTRODUCTION

User-produced multimedia data has grown abundant recently with the ubiquity of recording devices and online sharing websites, yet these files must be able to be retrieved easily for them to have value. Video detection systems that analyze and fetch multimedia files solve this problem of video event detection, or the classification of videos with regards to semantically defined events (e.g. marriage proposals or changing a car tire).

Preparing such multimedia classification systems often requires estimating the system's performance on unseen consumer-produced videos based on its performance on known consumer-produced videos. Fortunately, speculation of classifier performance is a well understood problem [7]. Extensive investigation of bootstrapping and its variations [5, 9, 4, 13] has identified bootstrapping as a prime, ableit not perfect [8], solution to this problem. We focus, however, on using bootstrapping to estimate the performance of specifically multimedia classifiers, and doing so efficiently for large applications.

When dealing with multimedia data, particularly given the increasing size of modern datasets such as the TRECVID MED 2013 dataset, efficiency becomes paramount. We turn to a recently developed bootstrapping method, the Bag of Little Bootstraps (BLB)[11, 12], that is designed to run quickly in a distributed setting on large amounts of data. An already existing DSEL compiler [3] converts input Python BLB applications into scalable BLB applications able to run on the Spark cluster computing system [14]. This DSEL compiler makes use of Selective Embedded Just-In-Time Specialization (SEJITS) [10], an approach for converting DSELs in productivity level languages to high performance efficiency language (e.g. C++ or Cilk) code.

Here, we code in this already existing Python DSEL a BLB application which estimates the standard deviation of the equal error rate (EER) of a video detection system. The generated distributed code efficiently predicts the performance on a subset of user-produced videos from the TRECVID MED 2013 dataset from performance on the Kindred dataset. We find that the predicted performance results generally fall in line with the error bounds obtained, demonstrating that despite high variability in consumer-produced videos, reasonable predictions regarding performance can still be made.

Finally, we further motivate usage of our methodology by comparing runtimes of large BLB applications with the original, serial Python code and the generated distributed code run on Amazon EC2 clusters. By demonstrating a difference of over two orders of magnitude between the different versions, it becomes clear that this methodology makes accessible an array of multimedia classifier estimation computations for those who never wish to leave the world of Python but previously were forced to.

2. CLASSIFICATION TASK AND DATA USED

Given the vast, expanding supply of user-produced multimedia videos, retrieving videos adhering to a particular description becomes essential in order to fully utilize large collections. Video detection systems that retrieve specified types of videos can be evaluated with a video event detection task in which they must determine whether videos belong to semantically defined events (e.g. wedding ceremony or making a sandwich).

The data used in the following experiments is a subset of the NIST TRECVID MED 2013 corpus, which comprises 160,000 consumer-produced videos of around three minutes each. For training, we select roughly 100 videos belonging to each of the 20 different "events," or classes, that range from changing a car tire to making a sandwich, and the 4,869 negative videos that belong to no event. Our primary test set consists of 26,399 files, of which 24,920 files do not correspond to any event while the number of positives for each event varies between 15 and 234. Additionally, we use the Kindred set, containing 14,249 files of which 12,770 are negatives, as a secondary test set. Before describing the experiments with this data, we describe in the next section a proposed methodology to accelerate the estimation of video detection systems.

3. METHODOLOGY DESCRIPTION

We propose a methodology to more efficiently estimate video detection systems' performance on event detection tasks; the methodology enables coding of bootstrapping applications to quantify uncertainty on performance estimates in Python, yet results in scalable, distributed code. When coding the bootstrapping application in Python, multimedia experts need only adhere to the specified DSEL for BLB, from which they must define with a modest subset of Python the statistical function to be estimated on the data set, along with the reducer function (e.g. standard deviation) that measures the error on the estimate. More detailed information about the DSEL, or the subset of Python allowed can be found in [3], but as an overview, all basic flow control statements (e.g. `for, while, if`), singularly-typed lists, variable assignments, arithmetic operations, basic type conversions, and simple list and string operations (e.g. `len, range, split`) are available.

For this paper's experiments, we choose to have the statistical function compute the EER of a multimedia classifier on a set of feature vectors, alternating between passing in the classification scores of each file, and computing them on the fly when given the models and extracted file feature vectors. The reducer function, which measures the error on the estimate, is the standard deviation. It would, however, be simple to modify the statistical function to estimate an alternate measure of classification performance, such as the misdetection rate at a certain false alarm rate. Similarly, the error estimate function could instead indicate uncertainty in terms of a confidence interval.

The already existing BLB DSEL compiler consumes the BLB Python application, and emits a scalable, distributed BLB application runnable on the Spark cluster computing system. Spark, similar to Map-Reduce, operates on clusters of commodity hardware, such as those purchasable with Amazon's EC2. Multimedia feature vectors, machine learning models, classification scores, or other input data is read from the Hadoop File System (HDFS) to support the large of multimedia datasets.

The Python code can similarly be mapped to backends besides Spark, namely OpenMP and Cilk, enabling efficient parallelism on a single node, further described in [3]. We do not utilize these alternate backends in the experiments because the less-general DSEL that they compile does not support the entirety of our applications. The pure Python code can additionally be run as unoptimized Python instead of being run on Spark, which is useful for debugging and small applications. This reusability of simple Python is convenient, since many bootstrapping applications, especially when file classification scores need not be computed, can run in a reasonable time in pure Python. However, for the larger applications as we later show, relying on the generated code is a must.

4. EXPERIMENTS

4.1 Predicting Performance on the MED Dataset

We estimate video detection system performance on a subset of the MED dataset from performance on the Kindred dataset. Classifying based on audio properties alone, we extract feature vectors from all videos using an i-vector system [6]. We then construct over 100 different training sets each containing the entirety of the roughly 100 training positives for each event but varying in the set of negatives used. The different negative sets are constructed from taking different representative samples from a k-means-clustering of the entire 4,869 training negatives provided. We next utilize SVMlight [2], an online support vector machine (SVM) library, to construct the training models for each of the training sets and to evaluate the models on the Kindred dataset in terms of EER, recording the best EER for each event.

To estimate the variability of the EERs, we run BLB 5 times on the Kindred dataset with 30 subsamples, 100 bootstraps, and a γ of 0.9 and average the results. We have empirically selected these parameters to ensure that the EERs from different BLB runnings match in the at least hundredths digit–most EERs obtained do agree to this standard with these parameters, but do not as well with reduced parameter sizes. The standard deviations obtained from BLB indicate the variation in EERs to be expected on datasets similar to Kindred, such as our primary test set which, although somewhat larger, contains videos that are sampled from the same underlying distribution, with even a slight overlap. To measure comparable performance on the primary test set, we again select a favorable training set in the manner described before (experimenting with the same different sets of training negatives) and record the EERs for the best training set.

4.2 Comparison to Naïve Python

We compare the performance of the generated Spark code to that of naïve Python, to motivate use of this methodology. We obtain runtimes for a BLB application that receives video classification scores and for a BLB application that receives video feature vectors and SVM models in order to compute the classification scores on the fly. In the previous section's experiments, there is no need to recompute the scores for each bootstrap, and so they can be passed in, but one can imagine classification applications where the file classification scores depend on the other files in the set. Both applications proceed to compute the classifier's EER for each event, and output the standard deviations that quantify uncertainty on the EER estimates. To simulate a "large" bootstrapping application, we duplicate the input scores (and feature vectors) from the primary test set five times over to create an input of 131,995 scores (or feature vectors), still tens of thousands of items less than the entire MED corpus contains.

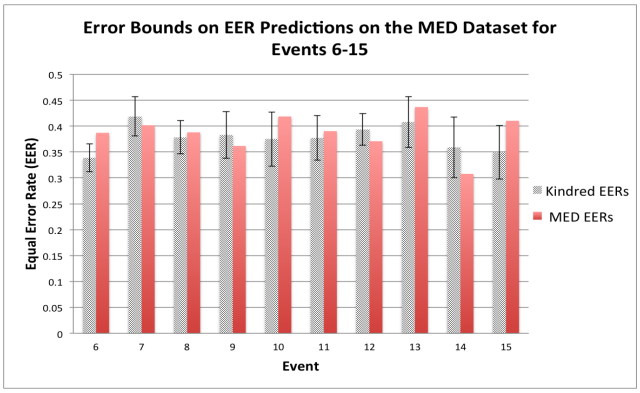
For the application receiving the scores, we again use parameters of 30 subsamples, 100 bootstraps, and a γ of .9. We run the Python version of the application on 1 Amazon EC2 [1] High-Memory On-Demand Instance (m2.4xlarge) and we run the distributed version of the application on 7 (1 master, 6 slave nodes) Amazon EC2 High-Memory On-Demand Instances (m2.4xlarge). For the application computing the scores, we use parameters of 2 subsamples, 5 bootstraps, and a γ of .9 (in light of the lengthy Python runtimes). We run the Python version of the application on 1 Amazon EC2 [1] High-Memory On-Demand Instance (m2.4xlarge) and we run the distributed version of the application on 13 (1 master, 12 slave nodes) Amazon EC2 High-Memory On-Demand Instances (m2.4xlarge). We further test the distributed version with parameters of 20 subsamples, 50 bootstraps, and a γ of 0.9 with the same number of nodes.

5. RESULTS

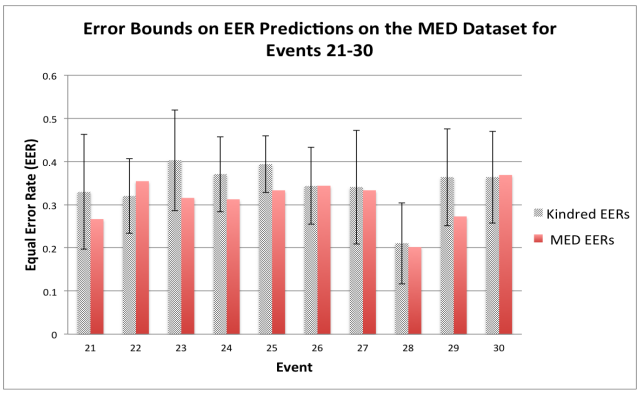
5.1 Predicting Performance on the MED Dataset

Figure 1 displays the EERs of the video detection system on both the Kindred and MED data sets along with the BLB predicted standard deviations for each event. The average EERs for the Kindred and MED datasets are 36.1% and 34.9%, respectively. The average predicted standard deviation is 7.2% and the average difference between event EERs for the Kindred and MED dataset in terms of predicted standard deviations for that event is .59. Interestingly, events 6-15 have an average number of 123.9 positives while events 21-30 have an average of 24 positives in the MED test set and this results in a significantly lower standard deviation of 4.3% versus 10.2% for the former. This indicates that high numbers of test positives may lead to more refined performance prediction estimates.

Although not all event EERs fall within one event standard deviation of each other, with the largest gap coming from event 6 with a 1.81 standard deviation gap, the performance on the MED dataset is reasonably bounded by the BLB's predictions. Put otherwise, the differences in EERs between the two sets, averaging 3.7%, correspond well to the scale of BLB standard deviations predictions, averaging 7.2%. This suggests that despite high acoustic variability and little structure, classification performance on consumer-produced multimedia files can be consistently predicted.



(a) Events 6-15



(b) Events 21-30

Figure 1: The performance of the video detection system on the Kindred and MED datasets. The error bars display the BLB standard deviations that estimate the variance on the MED dataset from the performance on the Kindred dataset.

5.2 Comparison to Naïve Python

As seen in Figure 2, the Spark code achieves more than a two order of magnitude speedup over the Python code, for both BLB applications tested. The Python code requires an average of 20681 seconds (5.74 hours) and 32142 seconds (8.93 hours) to run when given scores and when not, respectively, whereas the Spark code requires an average of 197.14 seconds (3.29 minutes) and 127.74 seconds (2.13 minutes) to run when given scores and when not, respectively. We can extrapolate runtimes for the BLB application that computes the scores for larger applications. Increasing the number of subsamples and bootstraps up to 20 and 50, respectively, results in 100 (10*10) times more estimates on bootstraps that need to be computed. A 100-fold increase in runtime would amount to a total runtime of 892 hours, likely longer than a researcher's patience would last.

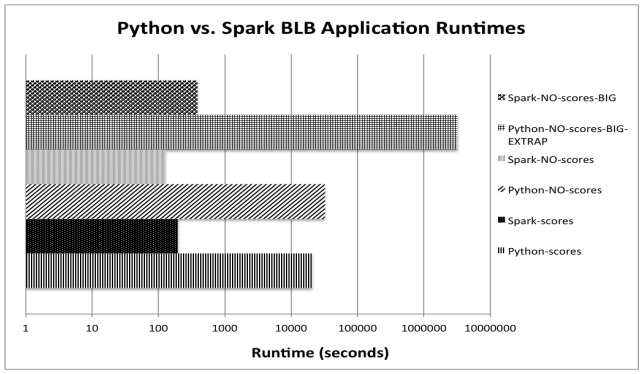


Figure 2: Comparison of runtimes of native Python and generated Spark code. The Spark code, finishing in less than 7 minutes achieves more than a 2 order of magnitude speedup over the 5+ hour Python times. The largest bar is the extrapolated runtime of Python on the BLB application that computes the scores with 100 times more bootstraps to compute than it we test with.

However, the generated Spark code completes in under 7 minutes, nearly 4 orders of magnitude better than the extrapolated Python runtime. The generated code performs better here relatively since the increased computational load supports a far greater degree of parallelism, whereas on the smaller computation with only 2 subsamples and 5 bootstraps, the need for only 10 bootstrap estimates severly caps the maximum paralellism. This trend of the generated code increasingly outperforming the Python code as the bootstrap computations needed and input data size increase would presumably remain true with more taxing applications since the generated code can simply run on more and more nodes to accomodate the higher demand.

6. FUTURE WORK

Future work involves further probing the user-produced video space with this methodology. This could involve examining the relationships between the numbers of training and test positives, the EERs, and BLB predicted standard deviations.

7. CONCLUSION

Our proposed methodology makes accessible to multimedia classification researchers coding in Python what was never accessible before: an efficient means to estimate video detection system performance via bootstrapping. This newfound accessibility to efficiency becomes critical on increasingly large consumer-produced multimedia datasets, as we demonstrate. With the aid of this methodology, the construction and improvement of video detection systems are made more feasible.

8. ACKNOWLEDGEMENTS

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates Nokia, NVIDIA, Oracle, and Samsung. Research also funded by DARPA Award Number HR0011-12-2-0016.

9. REFERENCES

[1] Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`.

[2] Multi-class support vector machine. `http://svmlight.joachims.org/svm_multiclass.html`.

[3] P. Birsinger, R. Xia, and A. Fox. Scalable bootstrapping for python. *ACM International Conference on Information and Knowledge Management*, 2013.

[4] M. Chernick, V. Murthy, and C. Nealy. Application of bootstrap and other resampling techniques: evaluation of classifier performance. *Pattern Recognition Letters*, 3(3):167–178, 1985.

[5] B. Efron. Bootstrap methods: another look at the jackknife. *The annals of Statistics*, pages 1–26, 1979.

[6] B. Elizalde, H. Lei, and G. Friedland. There is no data like less data: percepts for video concept detection on consumer-produced media. In *IEEE International Symposium on Multimedia ISM2013*. IEEE, 2013.

[7] K. Fukunaga and R. R. Hayes. Estimation of classifier performance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(10):1087–1101, 1989.

[8] A. Isaksson, M. Wallman, H. Göransson, and M. G. Gustafsson. Cross-validation and bootstrapping are unreliable in small sample classification. *Pattern Recognition Letters*, 29(14):1960–1965, 2008.

[9] A. K. Jain, R. C. Dubes, and C.-C. Chen. Bootstrap techniques for error estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):628–633, 1987.

[10] S. A. Kamil. *Productive High Performance Parallel Programming with Auto-tuned Domain-Specific Embedded Languages*. PhD thesis, EECS Department, University of California, Berkeley, Jan 2013.

[11] A. Kleiner, A. Talwalkar, P. Sarkar, and M. Jordan. The big data bootstrap. *arXiv preprint arXiv:1206.6415*, 2012.

[12] A. Kleiner, A. Talwalkar, P. Sarkar, and M. I. Jordan. A scalable bootstrap for massive data. *arXiv preprint arXiv:1112.5016*, 2011.

[13] B. Sahiner, H.-P. Chan, and L. Hadjiiski. Classifier performance prediction for computer-aided diagnosis using a limited dataset. *Medical Physics*, 35:1559, 2008.

[14] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.