Runtime Power Estimation for Mobile CPUs with Performance Monitoring Counters and Machine Learning

Eilleen Zhang[1,2], Onur Sahin[2], Victor Ly[2], Prof. Ayse K. Coskun[2]

Hillsdale High School, 3115 Del Monte St, San Mateo, CA 94403[1]

Electrical and Computer Engineering Department, Boston University, 8 St. Mary's Street, Boston, MA 02215[2]

Introduction

- Applications for mobile devices are becoming increasingly complex and power hungry, calling for improved energy-saving techniques due to limited battery capacity. Understanding power consumption in these devices requires accurate power estimation of mobile systems.
- In this project, we investigate how to utilize selected Performance Monitoring Counters (PMCs) and machine learning to predict power consumption of a mobile device during runtime.
- Performance Monitoring Counters (PMCs) are hardware counters that collect events from the processor and memory system during runtime.

Methodology

- We use the ODROID-XU3 mobile development board with ARM big.LITTLE core clusters.
- 8 cores total:
  - 4 LITTLE cores (A7 cores 0-3) maximize power efficiency
  - 4 big cores (A15 cores 4-7) maximize performance. We focus on the big cores because they consume significantly higher power than the smaller cores.
- A maximum of 6 PMCs can be collected simultaneously on the board while running a benchmark.
- Power is measured at the cluster level -- counters are measured at the per-core level.

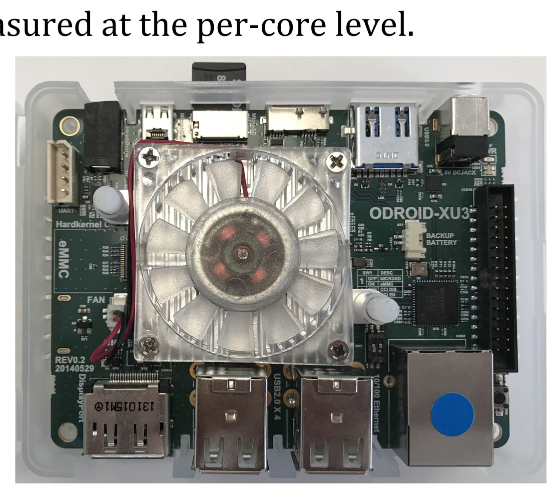


Figure 1 (right): ODROID-XU3 mobile development board used for data collection.

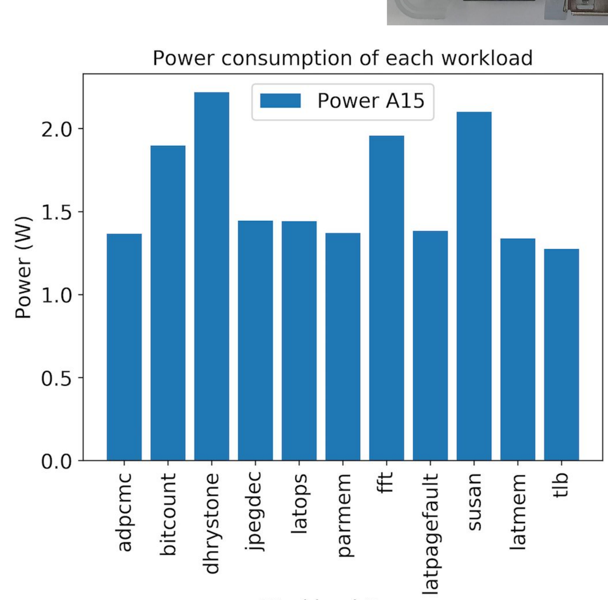


Figure 2 (left): We chose 11 benchmarks with varying power consumption levels.

Figure 3 (right): We refer to Walker et al[1] to choose 6 PMCs most correlated to power.

Event Hex	Event Name
0x11	CYCLE_COUNT
0x1B	INST_SPEC
0x50	L2D_CACHE_LD
0x6A	UNALIGNED_LDST_SPEC
0x73	DP_SPEC
0x14	L1I_CACHE_ACCESS
0x19	BUS_ACCESS

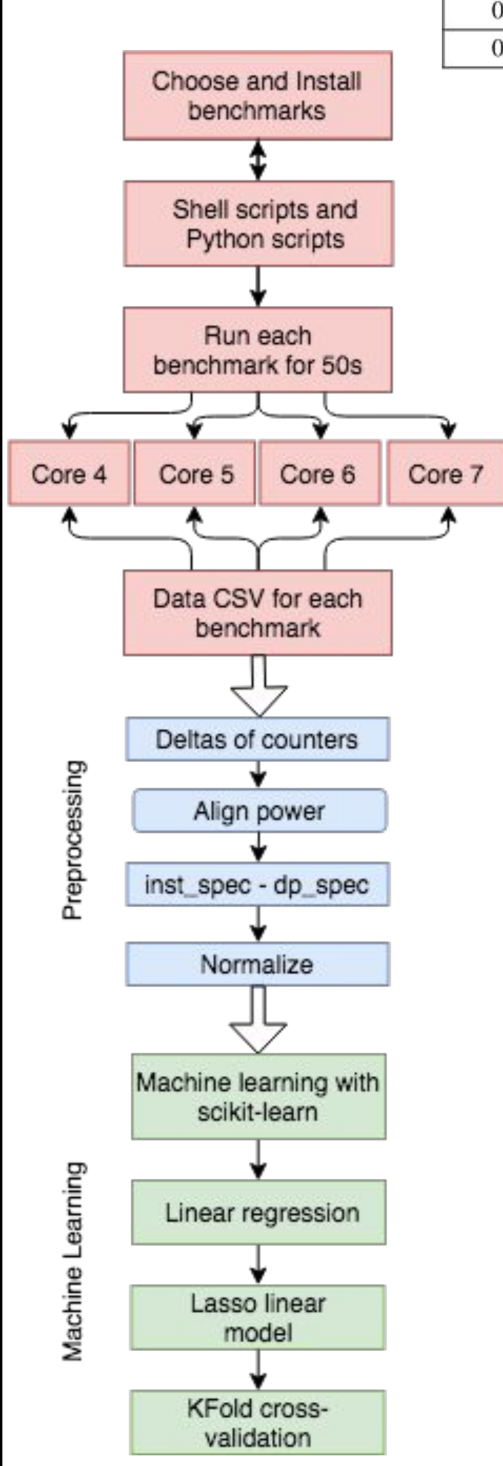
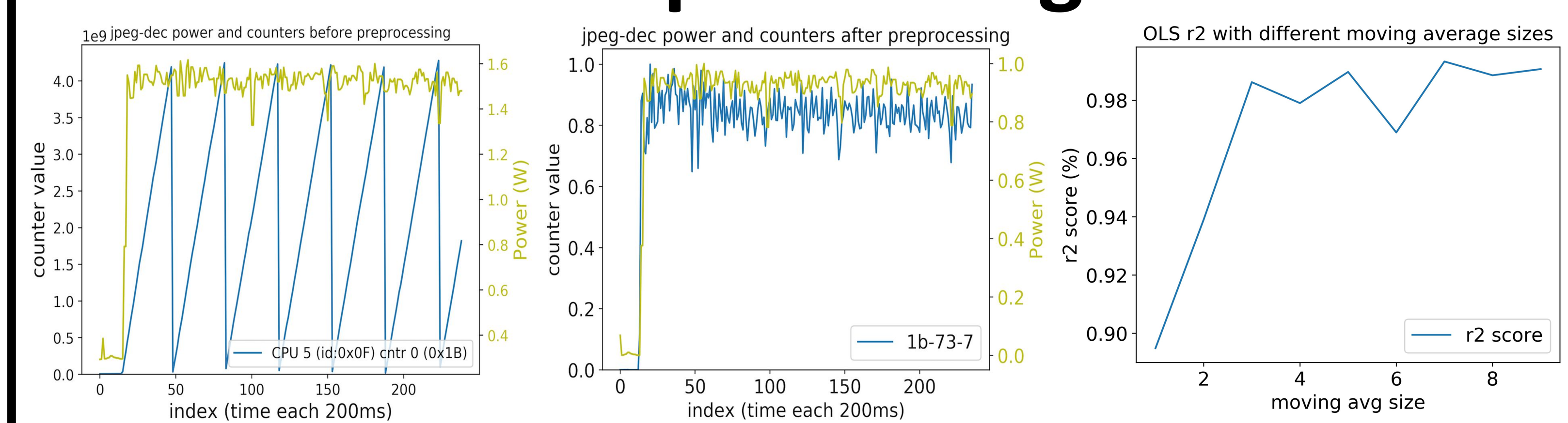


Figure 4 (above): Workflow for the project.

Our flow:

- Run benchmarks on the board using the android debug bridge (adb) shell.
- Make four copies of each benchmark
- Use taskset to assign the benchmark to cores 4-7.
- Collect data every 200ms and save the CSV files.
- Preprocess the data to prepare for machine learning with scikit-learn.
- Use machine learning to make a linear regression and find coefficients for each value.
- Check the accuracy with the r^2 score and mean squared error.
- Get the average of every 5 rows and save it as a separate training set.
- Use machine learning on the entire and averaged data sets.

Preprocessing

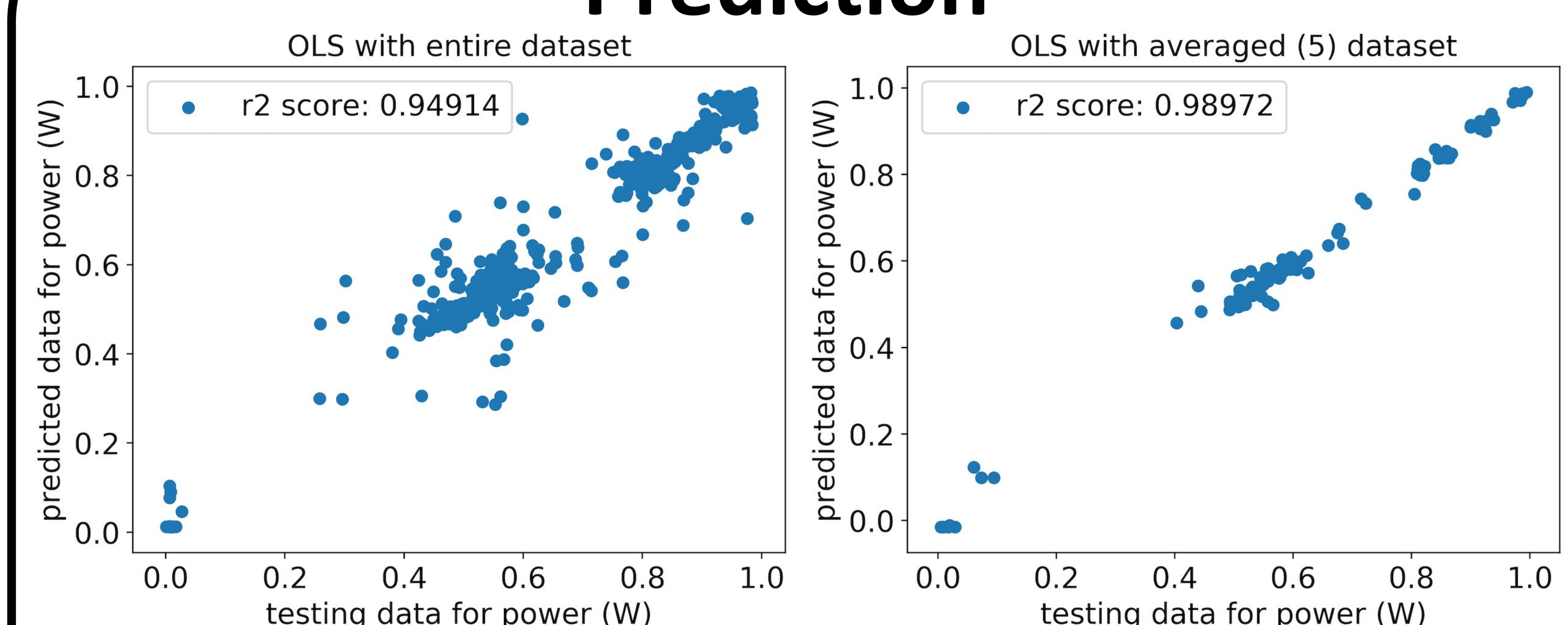


Preprocessing data for machine learning:

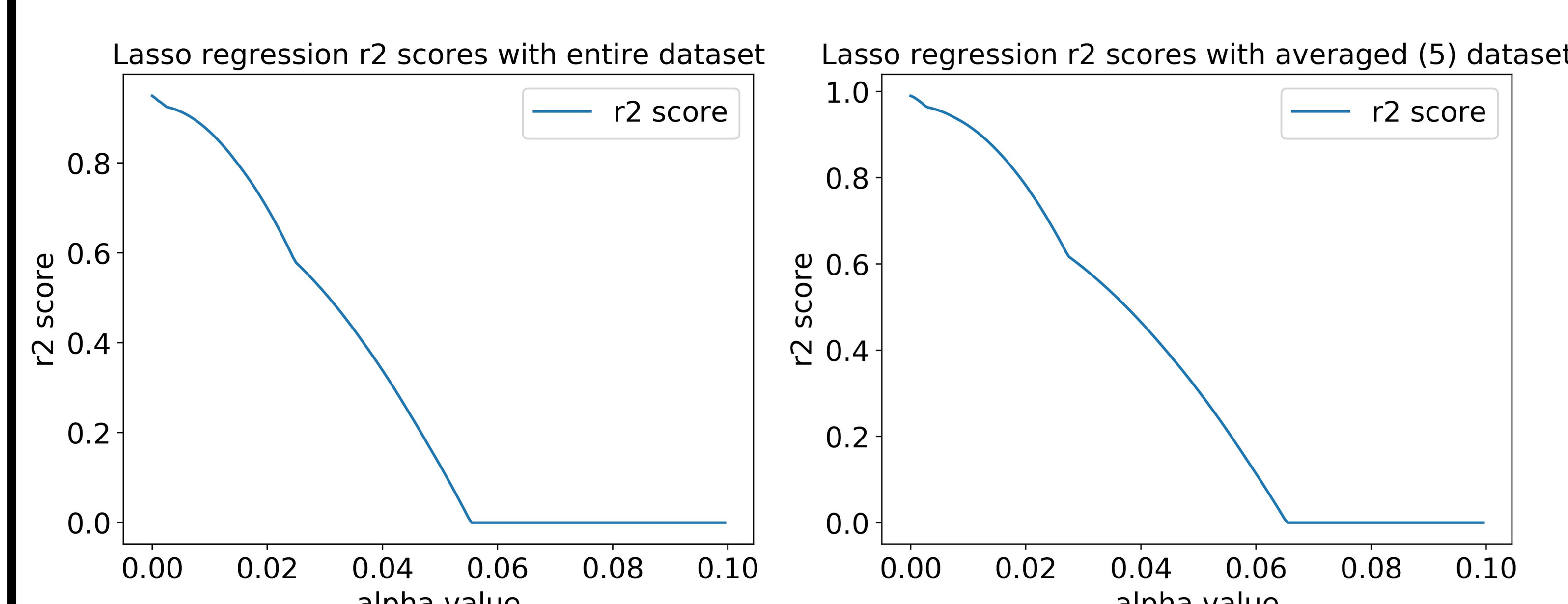
- Cumulative events → events per 200ms
- Separate values for inst_spec and dp_spec
- Normalize data to a range between 0-1
- Align power and events by initial spike
- Make a separate training set with the averages of every 5 data points

Figure 5 (left): Event 1B on core 5 for jpeg-dec workload, before preprocessing.
Figure 6 (middle): The same event after preprocessing.
Figure 7 (right): Comparing r^2 values when varying window size on averages, used to choose the window size for averaging.

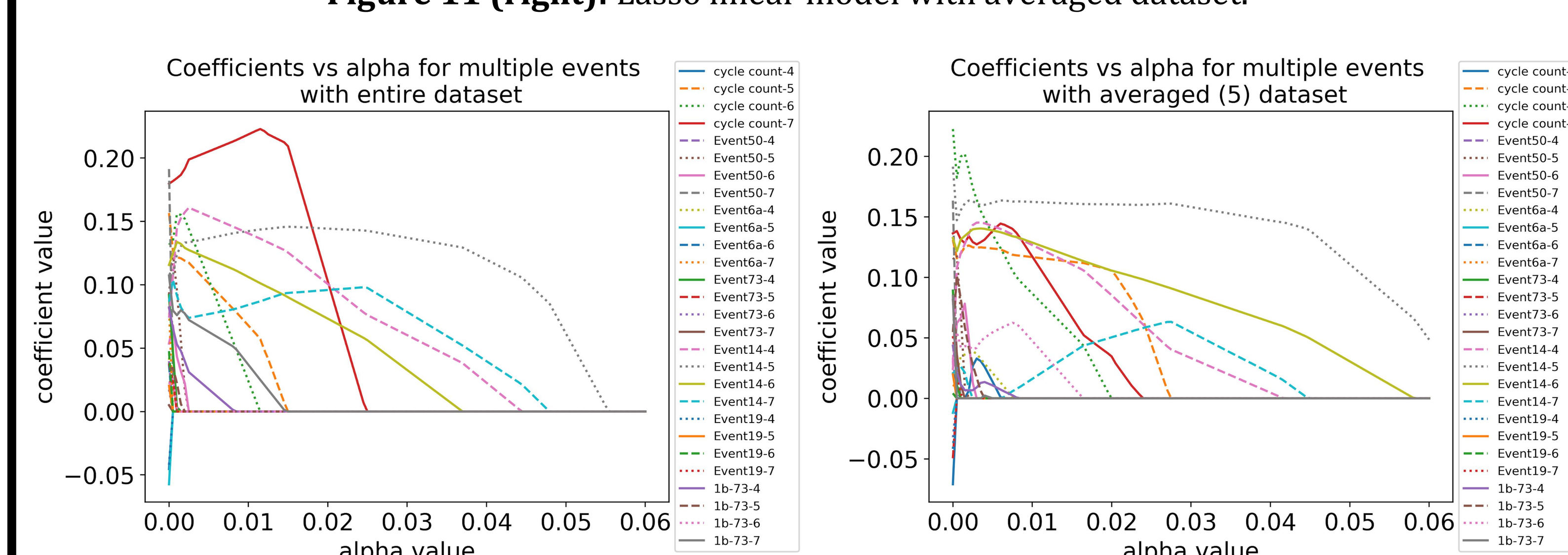
Prediction



Comparison of the linear regression, actual power vs. predicted power
Figure 8 (left): Prediction when using individual data points as training values.
Figure 9 (right): Prediction when using a moving average with window size 5.

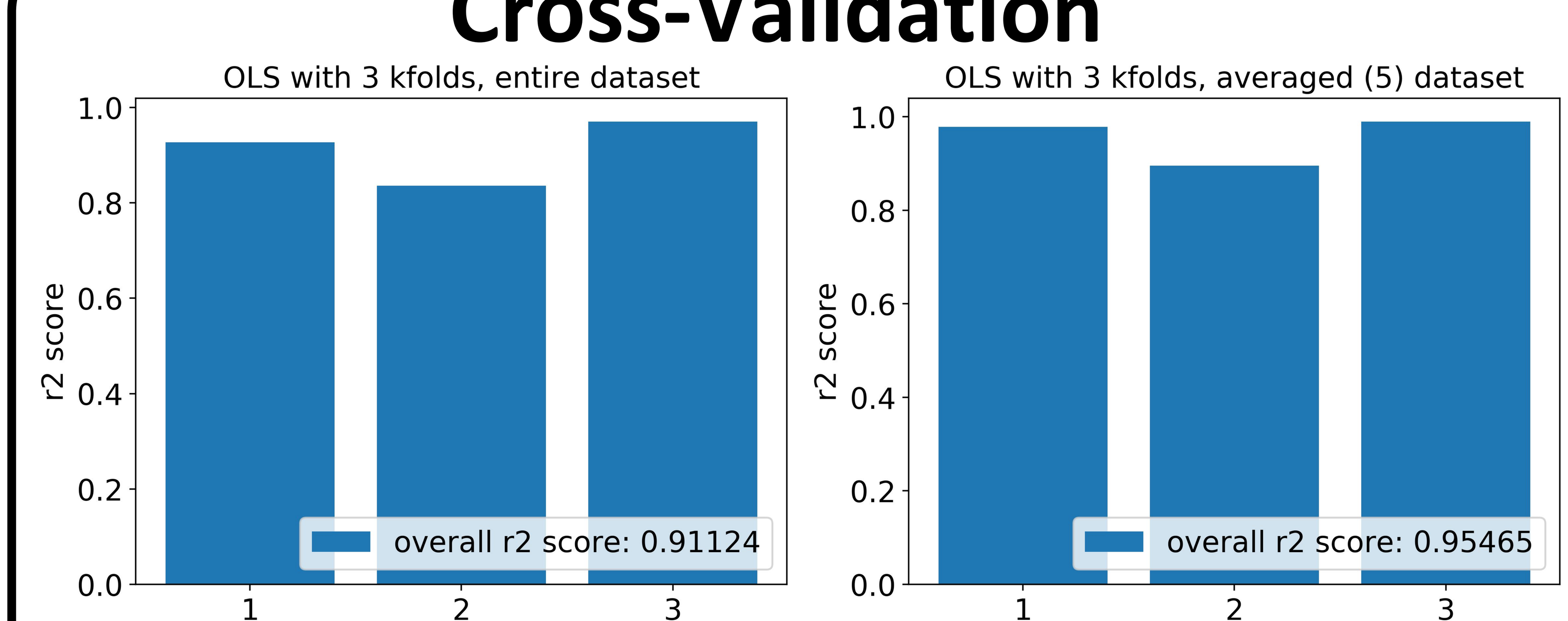


Lasso linear model used to reduce the number of coefficients in the linear regression. Graphs of r^2 values when using lasso on multiple alpha values.
Figure 10 (left): Lasso linear model with the entire dataset.
Figure 11 (right): Lasso linear model with averaged dataset.



Impact of varying alpha values on coefficients for each counter with Lasso.
Figure 12 (left): Comparing coefficients of multiple events with the entire dataset.
Figure 13 (right): Comparing coefficients of multiple events with the averaged dataset.

Cross-Validation



KFold cross-validation method used to analyze the machine learning model's stability.
Figure 14 (left): using 3 folds with linear regression on the entire dataset.
Figure 15 (right): using 3 folds with linear regression on the averaged dataset.

Machine learning with scikit-learn

- A Python machine learning library.

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

Figure 16 (above): Scikit-learn trains and tests linear models to find target value y. Each x is a feature and each w is a coefficient.[2]

- Ordinary Least Squares (OLS): a linear regression that minimizes the residual sum of squares between the predicted and actual power values.
- Lasso linear model: minimizes coefficients, examining the tradeoff between accuracy and reducing parameters.
- Accuracy is examined with the mean absolute error and the r^2 value.
- KFold: a cross-validation technique that examines the stability of the model.
  - separates the data into n number of "folds", trains the data with n-1 folds, and tests the data with the last fold. Repeat with other folds.

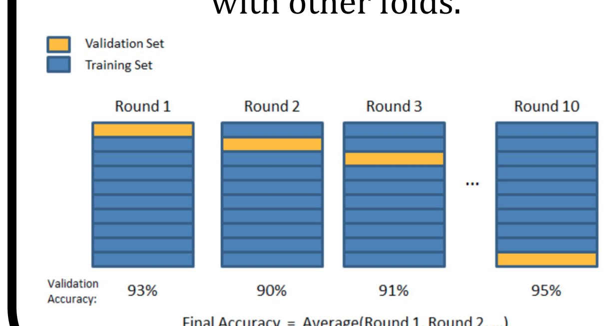


Figure 17 (left): How KFold splits a dataset based on the specified number of folds.[3]

Discussion

Conclusions:

- Power consumption can be modeled from these 6 PMCs with at least 91% accuracy.
- Using the the average of every 5 data points increases the accuracy to 98%.
- Lasso regression shows that certain PMCs with zero coefficients can be removed from the prediction without impacting accuracy.
  - The model is accurate up to an alpha value of 0.01

Caveats:

- Using the averaged data creates more extreme coefficients and increases the number of negative coefficients.
  - May be due to reduced size of dataset
- The lower r^2 score from 3 KFolds compared to OLS in Figure 8 and 9 demonstrates overfitting in the model. However, the accuracy remains at more than 90%.

Applications:

- These results demonstrate feasibility of predicting power consumption with more PMCs, and using Lasso to determine the most significant factors.

Future steps:

- Reduce overfitting with other fitting and cross validation methods
- Experiment with more linear modeling techniques from scikit-learn
- Experiment with a larger quantity of PMCs, using Lasso to determine the most important features in power prediction

References

[1] Walker, M. J.; Diestelhorst, S.; Hansson, A.; Das, A. K.; Yang, S.; Al-Hashimi, B. M.; Merrett, G. V. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2017**, *36*(1), 106–119.

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

[3] Bronshtein, *A. Train/Test Split and Cross Validation in Python Towards Data Science*, https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6 (accessed Aug 6, 2018).

Acknowledgements

Thanks to Victor Ly, Onur Sahin and Prof. Coskun, as well as PEAClab group for their support and guidance throughout this project. Thanks to the Boston University RISE Internship Program for giving me the opportunity to do research in a lab setting.