HOW MUCH MEMORY RADIATION PROTECTION DO ONBOARD MACHINE LEARNING ALGORITHMS REQUIRE?

Kiri L. Wagstaff and Benjamin Bornstein

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA, 91109, USA

ABSTRACT

Onboard autonomy is necessary for achieving the goals of future space missions, given the communication delays imposed by the extreme distances involved (e.g., to Saturn). The high-radiation space environment can have severe negative effects on unprotected onboard computation, for example by flipping bits in memory. Radiation-hardened components that protect against the majority of these errors exist, but whether such extreme protection is needed is an open question. We developed a method for simulating radiation-induced bit flips and quantitatively assessed the sensitivity of clustering and classification algorithms likely to be used onboard spacecraft. We found that, for small data sets in a low-Earth orbit radiation environment, commercial RAM would suffice; no radiation-hardening of the memory is needed. We also found that simpler algorithms (regular k-means clustering, linear support vector machines) have less sensitivity (more tolerance) than more sophisticated versions (kd-k-means, Gaussian support vector machines). The development of algorithms with even less sensitivity to radiation is an open area of research.

Key words: radiation, onboard data analysis.

1. INTRODUCTION

Space missions are adopting increasingly ambitious goals, in which they operate at distances and in environments that require onboard autonomy for local decision making. Autonomy that closes the decision loop without resorting to communications with the Earth can dramatically reduce idle time and increase mission capability and science return. Proposed future missions to the Jupiter and Saturn systems, such as the Jupiter Europa Orbiter or a Titan aerobot, stand to benefit significantly, since each one-way communication consumes hours instead of minutes (as with Mars). Onboard machine learning and data analysis methods are what can make this level of autonomy possible. They enable spacecraft to make decisions about how to prioritize data, the level of compression to use when transmitting data, and which targets to observe next (Castano et al., 2005, 2007). The primary methods used are unsupervised clustering and supervised classification. Clustering identifies groups and trends in the observations and can also be used for anomaly detection. Classification can serve science goals (e.g., to identify land cover types in a hyperspectral image) or engineering goals (e.g., to distinguish normal from abnormal operations).

The onboard computing environment differs from a desktop environment in many ways, including the amount of computational power, available memory, storage space, and high doses of radiation. Radiation causes errors in the processor (corrupting registers and caches) and in memory (flipping individual bits). The former problem is generally addressed by using fully radiation-hardened processors. We focus on the effects of radiation on onboard memory and the subsequent impact on the results of onboard data analysis. Spacecraft currently employ both hardware and software countermeasures to address the radiation problem for memory. The hardware solution involves the use of radiation-hardened components, which bring with them a steep increase in cost, sometimes an increase in mass, and almost always a reduction in capability and speed, compared to less-hardened components. Existing software solutions include EDAC (Error Detection and Correction), which employs a "memory scrubber" process to continually check memory for errors (Shirvani et al., 2000). It can detect and correct single-bit errors, and detect (but not correct) double-bit errors. However, in addition to the processing overhead imposed by the memory scrubber, EDAC also imposes a memory overhead of about 30% due to the additional bits needed to permit the detection of a flipped bit.

Intuitively, we would like to prevent onboard computation from suffering any radiation-induced errors. However, it is quite possible for an algorithm to produce the correct results even in the presence of the occasional flipped bit, especially if the affected bits are not needed for the final calculation or are flipped after they are read for the last time. If so, then the protection afforded by radiation-hardened memory and EDAC systems would be unnecessary, and spacecraft could employ higher-capability, cheaper, lighter, and less-hardened memory for onboard computation.

We therefore propose a quantitative measurement of the innate degree of radiation tolerance of a variety of on-

Proc. 'IJCAI–09 Workshop on Artificial Intelligence in Space', Pasadena, California, US 17–18 July 2009 (ESA SP-673, September 2009)

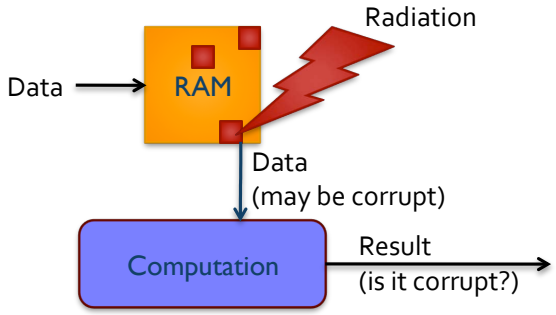


Figure 1. The BITFLIPS software radiation simulator injects errors (single-event upsets or SEUs) into RAM while the program is running. When the algorithm requests data from RAM, it may be corrupted.

board data analysis algorithms. To do this, we assess an algorithm's performance in the presence of simulated radiation-induced memory errors. These "single-event upsets" (SEUs) are simulated for the input data that is being analyzed as well as any intermediate data structures the algorithm creates while operating. We developed a software radiation simulator called BITFLIPS (Basic Instrumentation Tool for Fault Localized Injection of Probabilistic SEUs). BITFLIPS monitors all data structures in use and injects simulated SEUs at a user-specified rate (in SEUs per kB per instruction).

The major contributions of this paper are 1) the BITFLIPS software radiation simulator and 2) radiation sensitivity results for both classification and clustering algorithms that are, or may in the future be, used onboard spacecraft. The same radiation sensitivity evaluation methodology can be used with other algorithms, input data, and specified radiation environments so that it is possible to determine in each case whether less-hardened memory can be safely employed. More generally, this kind of analysis allows mission designers to select individual algorithms according to the target radiation environment. For the first time, the merits of different algorithms, with respect to their radiation tolerance, can be directly compared.

2. RADIATION SIMULATION: BITFLIPS

While radiation can cause errors both in spacecraft memory and in the processor, we focus on modeling radiation impacts in memory only. The CPU is such a critical component to the entire spacecraft, not just the data analysis system, that it is likely to be fully radiation-hardened for the foreseeable future. However, spacecraft memory could potentially tolerate less hardening, if the software itself is shown to be tolerant to the target radiation environment. The use of less-hardened memory components could greatly decrease the cost and increase the capability of a mission. Further, even radiation-hardened memory experiences the occasional error, so characterizing a given algorithm's sensitivity to radiation is relevant regardless. Radiation can cause a variety of errors in memory, include flipped bits, stuck bits, and damaged components. The most common error is the single-event upset (SEU).

2.1. Simulating SEUs in BITFLIPS

We designed and implemented a lightweight SEU software simulator, BITFLIPS (Basic Instrumentation Tool for Fault Localized Injection of Probabilistic SEUs), that is built on the Valgrind debugger/profiler (Nethercote and Seward, 2007). BITFLIPS injects errors in a reproducible fashion at a user-specified SEU rate (see Figure 1). For programs written in C, macros are provided that enable the selective exposure (and protection) of individual program variables.

The open source Valgrind debugging and profiling tool provides an ideal foundation for BITFLIPS. Valgrind simulates a CPU in software and provides a modular architecture for creating tools that hook into its simulation environment. Valgrind's stock tool suite contains a memory leak detector, CPU cache profiler, program caller-callee inspector, system heap profiler, and a thread synchronization debugger. BITFLIPS is patterned after Valgrind's memory leak detector, but instead of monitoring memory usage, BITFLIPS injects SEUs into memory during program execution.

BITFLIPS relies on Valgrind's on-the-fly program instrumentation capability to inject SEUs. To simplify instrumentation, Valgrind translates a program's processor specific instructions into VEX IR, a Reduced Instruction Set Computing (RISC)-like Intermediate Representation (IR) language. RISC-like instructions eliminate the need for plugin tools like BITFLIPS to contain specialized program logic tailored to complicated, possibly processor-specific instructions. Instead tools analyze and operate on basic load, store, arithmetic, comparison, and branch operations. The Valgrind simulator, and by extension, BITFLIPS, operates in an instrument-execute loop.

The instrumentation process begins when Valgrind translates the first (or next) block of a program's processor specific instructions into VEX IR. Next, Valgrind passes its VEX IR block to BITFLIPS for analysis and instrumentation. BITFLIPS then interleaves a special C-callback VEX IR instruction between each of the program's VEX IR instructions in the block. The callback instruction, when executed, results in a call to a BITFLIPS C function, `BF_doFaultCheck()`, which is responsible for deciding when and where to inject SEUs. The `BF_doFaultCheck()` function delegates to `BF_doFlipBits()` when appropriate to perform the actual SEU operation. When BITFLIPS finishes its instrumentation, the VEX IR block is passed back to Valgrind. Finally, Valgrind executes the instrumented instruction block. This process repeats until there are no more program instructions left to execute.

The rate at which BITFLIPS injects SEUs is governed by a radiation flux parameter which is fixed at the time of initial program execution. The units of this parameter are SEUs per kilobyte per instruction. We use kilobytes as a proxy for physical memory area; the larger the area, the more memory is exposed to radiation. The SEU density (number of bits flipped per SEU) is determined by a discrete Poisson distribution. Sixty percent of BITFLIPS' SEUs affect a single bit. Thirty percent of BITFLIPS' SEUs affect two bits, and so on. An upset affecting seven bits is exceedingly rare and accounts for only one percent of SEUs injected.

2.2. Exposing Individual Data Structures

For both precise experimental control and improved reporting, BITFLIPS allows the specification of which program variables to expose to radiation. There are two requirements for this capability: 1) the program must be written in C and 2) the program source code must be accessible for compilation. Exposing (or shielding) individual variables is achieved through a Valgrind feature known as Client Request Macros (CRMs). Valgrind CRMs are C preprocessor macros whose substituted code results in a series of register bit shifts. When a program is run under Valgrind, these register operations are detected by the Valgrind simulator and mapped to C callbacks in BITFLIPS. When a program is run in its native environment (i.e., outside of the Valgrind simulator), the register operations are effectively no-ops, and they do not affect the operation of the program. Moreover, the run-time overhead imposed by CRMs on native programs is negligible: six simple integer instructions. BITFLIPS CRMs also communicate to BITFLIPS the C type (e.g., `char`, `int`, `float`, `double`, etc.) and layout (for row- or column-major matrices) of the exposed program variables. BITFLIPS uses this information, in its verbose output mode, to report variable values before and after an SEU, as well as the difference between them, so that the magnitude of the SEU's impact can be quantified.

The BITFLIPS CRMs are as follows:

- `VALGRIND_BITFLIPS_ON()` Enables SEU injection.
- `VALGRIND_BITFLIPS_OFF()` Disables SEU injection.
- `VALGRIND_BITFLIPS_MEM_ON(addr, nrows, ncols, type, order)` Exposes a block of memory beginning at address *addr* to SEUs. The memory block has *nrows* rows and *ncols* columns. The C *type* of the block (e.g. `char`, `int`, `float`, `double`, etc.) and matrix *order* (row- or column-major) give BITFLIPS additional information to use when reporting variable values before and after an SEU.
- `VALGRIND_BITFLIPS_MEM_OFF(addr)` Shields the previously exposed block of memory beginning at address *addr* from future SEUs.

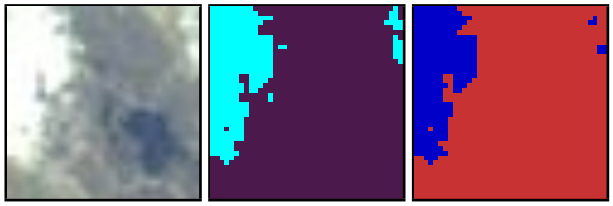
3. CLUSTERING ALGORITHMS

Our first experiments were with clustering algorithms. An initial version of these results was previously reported by Wagstaff and Bornstein (2009). However, an incorrect conversion from the native SEU rate used by BITFLIPS (in SEUs per kB per instruction) to SEUs per kB per second caused an incorrect conclusion regarding effective radiation rates in low-Earth orbit (LEO). We reported that, for small data sets, the effective rates experienced by commercial RAM in LEO would not cause trouble for the analysis of small data sets, but would be problematic for larger data sets. This would be true if the processor were only capable of executing a single instruction per second. Instead, for modern processors (even radiation-hardened processors), which can execute millions of instructions per second, in fact it would be safe to cluster orders of magnitude more data, using commercial RAM, without seeing any adverse effects in the LEO environment. Revised figures that use the correct version of radiation rates used by BITFLIPS, and our revised conclusions, are presented in this section.

3.1. Clustering Methodology

We compared the basic k-means clustering algorithm (MacQueen, 1967) to two variants designed to speed up analysis. A shorter runtime would mean less exposure to SEUs and therefore potentially higher radiation tolerance. Subsampling k-means (Bradley and Fayyad, 1998) performs an initial pass with a subset of the data to obtain a good starting point for clustering the whole data set, yielding faster convergence. Kd-k-means (Alsabti et al., 1998; Kanungo et al., 1999; Pelleg and Moore, 1999) builds a kd-tree over the data's feature space and uses this to "filter" cluster centers to the correct data points. It achieves faster runtime at the expense of consuming more memory, to store the kd-tree.

We experimented with two data sets. The Iris data set from the UCI repository (Asuncion and Newman, 2007) contains 150 items, each described by four features. There are three types of iris in this data set, so we set the number of desired clusters $k = 3$. The second data set consists of 1600 pixels (40x40) collected by the Hyperion instrument onboard the EO-1 Earth orbiter on October 3, 2002. Each pixel is described by 11 features (reflectance at 11 different wavelengths, from 426 to 2284 nm). The data set covers part of the Quinghai Province in China and includes clouds, cultivated land, and lakes as shown in Figure 2(a). For evaluation purposes, we manually labeled each pixel as "cloud" or "not cloud"; see Figure 2(b). We set $k = 2$ when clustering and obtained



(a) RGB (b) Labels (c) K-means

Figure 2. Satellite image of Quinghai Province, China, observed on October 3, 2002. (a) The 11-band data shown in RGB. Clouds are on the left and right sides, with land and a lake between them. (b) Manually assigned labels: bright = cloudy and dark = clear. (c) Regular k-means clustering output (using all 11 spectral bands, no radiation) achieves 0.940 agreement with the labels.

the result in Figure 2(c) from the k-means algorithm when clustering in a radiation-free environment.

We applied each of the three algorithms to each data set with a variety of different radiation rates specified. We evaluated performance using the Adjusted Rand Index or ARI (Hubert and Arabie, 1985), which reports the amount of agreement between the clustering output and the true partition defined by the data labels. The maximum possible ARI score is 1.0.

3.2. Clustering Results

Our first finding was that using subsampling with k-means measurably improves its radiation tolerance, as shown in Figure 3. At a radiation rate of 1×10^{-5} SEUs/kB/inst, performance for k-means drops from an average of 0.7 down to 0.32, while subsampling k-means remains at 0.64. At higher radiation rates, the performance of all algorithms drops to 0. In contrast, the use of a kd-tree worsens radiation tolerance. Performance begins to decline at a much lower rate of 8×10^{-7}. This is largely due to the increased memory profile (storing the kd-tree itself) and the algorithm's sensitivity to any perturbations to this tree.

For comparison, the Mongoose-V processor on the EO-1 spacecraft (in low Earth orbit) can execute about 20 MIPS and, using regular commercial RAM, would experience about 4.6×10^{-14} SEUs/kB/inst. All of the radiation rates shown in Figure 3 are therefore *much* higher than anything such a spacecraft would experience in low Earth orbit, and we could safely use commercial RAM when analyzing this data set, or even data sets that are orders of magnitude larger. In a higher radiation environment, such as Jupiter orbit, this may no longer be the case. Given estimates of the SEUs/kB/sec experienced in that environment, we can conduct the same analysis and determine how much radiation hardening is needed for the spacecraft memory.

The BITFLIPS simulator allows us to selectively expose

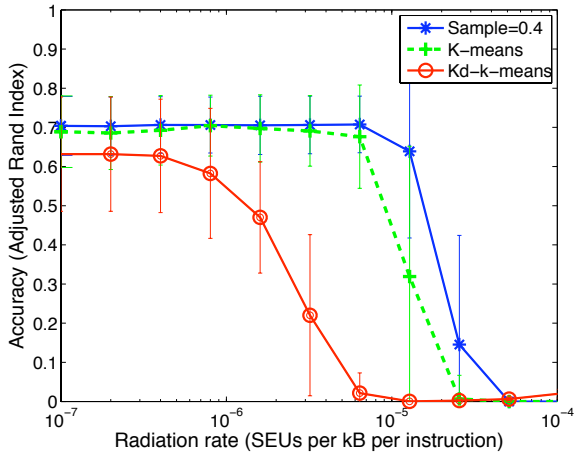


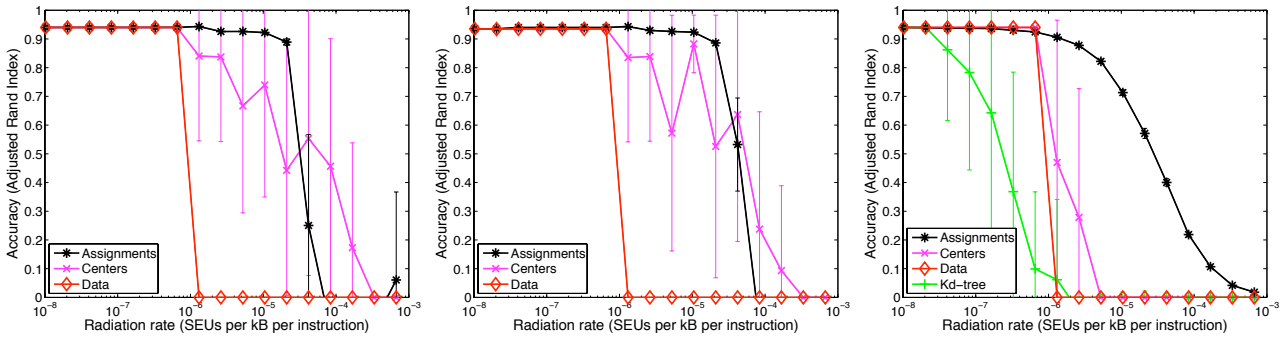
Figure 3. Radiation tolerance of three k-means clustering algorithms, with all RAM exposed, measured as clustering accuracy as a function of radiation rate. Results were obtained by clustering the Iris benchmark data set and averaging over 100 trials.

different parts of memory to radiation, so that we can precisely probe and identify which data structures are most vulnerable to radiation, and therefore may require the most protection. Figure 4 shows the performance results obtained when clustering the Quinghai Province satellite data using each of the three clustering algorithms, and exposing either the input data, the assignments of items (pixels) to clusters, or the cluster centers themselves (the cluster models). In addition, for kd-k-means, we tested exposing only the kd-tree.

We found that radiation sensitivity was dictated by how the memory was used by the algorithm, rather than the raw amount of memory that was exposed. We consistently found for all algorithms that the cluster assignments were the least sensitive component of memory usage, even though they consumed more memory than the cluster assignments (for example) did (6400 versus 88 bytes). In addition, we found that the kd-tree itself was the most sensitive data structure used by the kd-k-means algorithm, helping to explain its increased sensitivity when all data structures were exposed, as in Figure 3. However, once again the SEU rates at which degradation was observed were 8 orders of magnitude larger than what would be experienced by the EO-1 spacecraft that collected this data, even if it used only commercial RAM.

4. CLASSIFICATION ALGORITHMS

Classification algorithms are also of interest for onboard use, particularly Support Vector Machines (Cortes and Vapnik, 1995), which are currently used onboard the EO-1 orbiter (Castano et al., 2005). These algorithms are first used to train a classifier model on the ground, using labeled data, and then the learned model is uploaded to the



(a) Regular k-means (b) K-means with 10% subsample (c) Kd-k-means (leaf size = 100)

Figure 4. Radiation tolerance of individual data structures used by k-means algorithms (Quinghai Province data set). Each curve is the average over 10 trials, with bars showing one standard deviation.

spacecraft and used to classify new observations as they are collected. On EO-1, the classification result can be used to autonomously trigger additional observations of key phenomena such as lake ice thaw or the appearance of arctic sulfur deposits.

4.1. Classification Methodology

Support vector machines (SVMs) rely on a *kernel function* to specify how the similarity between two items should be measured. The simplest kernel function is the dot product between two items, which is referred to as a linear kernel. It is effective for learning to classify items when the classes are linearly separable. If not, more sophisticated kernels using polynomial and Gaussian (RBF) functions are employed. These kernels, while capable of capturing more complicated class boundaries, are more expensive to compute. Their longer runtimes may therefore make them more susceptible to radiation.

The data set we used for the classification experiments was a synthetic data set composed of 1000 six-dimensional items artificially generated from two Gaussian distributions $\mathcal{N}_1$ and $\mathcal{N}_2$. The means of these two distributions were randomly generated from $\mathcal{N}(0, 1)$, and each had a standard deviation of 1. The resulting two classes were not linearly separable.

We experimented with different kernels to determine their relative sensitivity on this data set. We evaluated performance using the classification accuracy of the learned model on a separate test set of 200 items (100 from each class). The SVMs were always trained in the absence of radiation, akin to how they would be trained "on the ground" for a real deployment. We simulated SEUs during the testing phase, which corresponds to the onboard use of the trained model. We also experimented with selectively exposing the input data, the learned model, or both. Each trial varied the random seed provided to BITFLIPS, creating a different pattern of SEUs in RAM, while keeping the trained model and input data fixed.

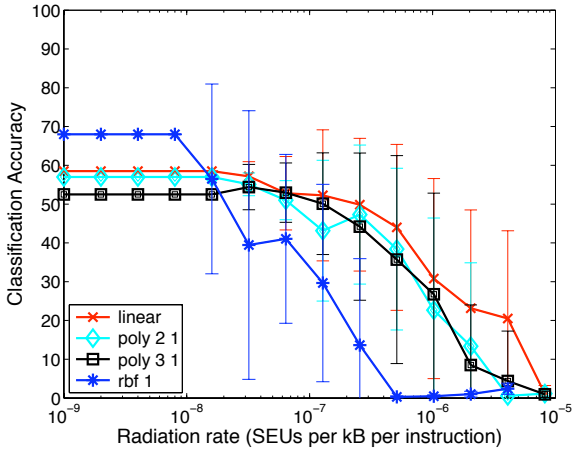
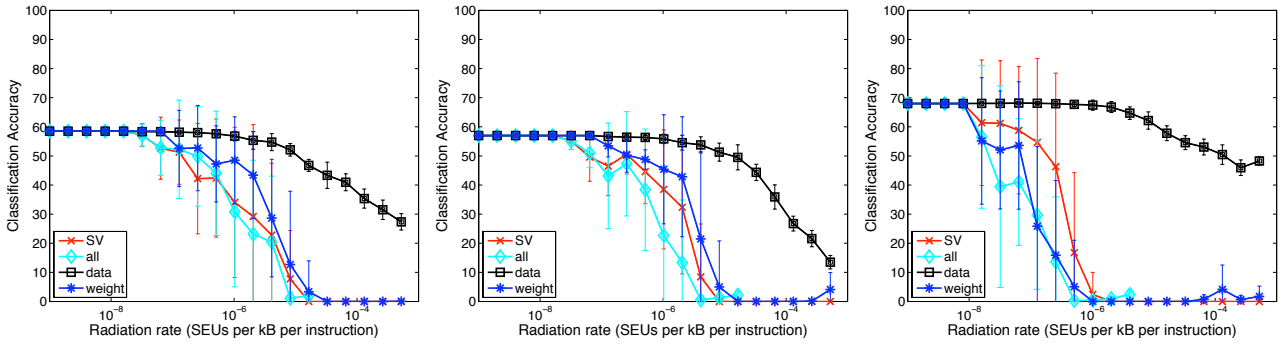


Figure 5. Radiation tolerance as a function of radiation rate for support vector machines using different kernels, with all RAM exposed, averaged over 10 trials. All SVMs were trained on 1000 items from two Gaussian distributions and tested on 200 additional items.

4.2. Classification Results

Figure 5 shows that, at very low radiation rates, the Gaussian (RBF) SVM had the best performance, classifying 68% of the test items correctly. The linear SVM had an accuracy of only 59%, and the second- and third-degree polynomial kernels yielded even worse performance. However, the Gaussian SVM's performance fell off quickly as the radiation rate increased, probably because of its longer runtime (an order of magnitude longer than a simple linear kernel). In fact, the simple linear SVM showed the best radiation tolerance, retaining the best performance at higher radiation rates (albeit with large error bars).

The radiation tolerance results when exposing individual data structures used by the SVMs are shown in Figure 6. A trained SVM model consists of a collection of support vectors (SVs) and the weight associated with each support vector. These are collectively used to classify



(a) Linear kernel (b) Second-degree polynomial kernel (c) Gaussian (RBF) kernel

Figure 6. Radiation tolerance of individual data structures used by support vector machines, as a function of radiation rate. All SVMs were trained on 1000 items from two Gaussian distributions and tested on 200 items, and results were averaged over 10 trials.

new items. We tested selectively exposing the SVs, the weights, the input data, or all of the above (the "all" results are identical to those shown in Figure 5).

Interestingly, the relative sensitivity of different data structures for SVMs was the opposite of what we observed with clustering algorithms. For clustering, the input data (and the kd-tree, if used) was the most radiation-sensitive area of memory. For SVMs, the data was the *least* sensitive component. The difference arises from how the algorithms work. K-means (and its variants) use an iterative strategy, assigning items to clusters, updating the cluster models, and then repeating the process until the solution converges. Errors in the data persist and affect every subsequent iteration. In contrast, an SVM examines each data item only once, generates its classification result, and moves on to the next item. SEUs that occur in items that have already been classified, therefore, have no effect on the final outcome. Likewise, the model created by k-means clustering (the cluster means) is less sensitive to radiation because it is refreshed at every iteration; each cluster mean is recomputed from the items assigned to it. But the model used by an SVM is not re-trained or updated during classification, so again any errors are cumulative.

This inversion emphasizes the fact that, if selective decisions must be made about which memory to protect and which may use less hardened components, it is critical to consider what algorithms will be run and their specific vulnerable points in terms of the individual data structures that they use.

Curiously, we observed that the relative sensitivity of the components of the SVM model (the support vectors and the weights) was not constant across different kernel types. For a linear kernel (Figure 6(a)), the SVs were more sensitive than were the weights, but this difference was reduced for the second-degree polynomial kernel (Figure 6(b)), and inverted for the Gaussian kernel (Figure 6(c)), for which the weights were more sensitive than the SVs. There is no immediately obvious explanation for this phenomenon, and it could be an effect of running only 10 trials, with a single model for each kernel, in which case the distinction would vanish if the results were averaged over a broader variety of runs.

5. RADIATION PROTECTION FOR ONBOARD MACHINE LEARNING ALGORITHMS

For the data sets we the have used to date, we found that, for operation in low Earth orbit, even commercial RAM provides more than enough protection for both classification and clustering algorithms to produce correct results despite ongoing SEUs. However, in higher-radiation environments, such as Jupiter or Saturn orbit, or when analyzing larger data sets that necessarily require long runtimes, we may reach the point where commercial RAM does not suffice. In such a case, radiation-hardened memory could be used. However, due to the negative factors previously mentioned, it is worth considering what software options may be available that would provide the same level of reliability.

There are three main software approaches to providing radiation protection. As mentioned before, Error Detection and Correction (EDAC) uses special memory encoding schemes paired with a periodic "memory scrubbing" process to detect and correct SEUs as they happen (Shirvani et al., 2000). Second, one can pre-emptively execute the same computation (such as classifying items) multiple times and combine the results with a voting scheme. EDAC imposes an overhead in terms of memory storage (about 30%) and processor consumption (for the scrubber process). Pre-emptive execution may waste cycles recomputing unnecessarily. Algorithm-Based Fault Tolerance (ABFT) addresses these limitations by creating algorithm-specific checks that allow the detection of errors in the output (Huang and Abraham, 1984; Turmon et al., 2003; Granat et al., 2009). There is a small additional effort involved in conducting these checks, but it allows the system to only recompute solutions that contain errors, and does not require any special memory encodings or a monitoring process to run in the background.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a software radiation simulator (BITFLIPS) which allows the precise testing of an algorithm's radiation sensitivity, or tolerance. BITFLIPS allows the specification of a radiation rate, in terms of SEUs per kilobyte of RAM per instruction executed by the CPU. It also permits the selective exposure of individual data structures in memory, so that the most vulnerable portions of an algorithm's memory usage can be identified.

We evaluated the sensitivity of both clustering and classification algorithms. These are algorithms that are currently, or are likely to be, used onboard spacecraft in high-radiation environments. We found that in both cases, for small data sets, performance did not degrade until the radiation rate was increased by 8 or 9 orders of magnitude above that experienced in low-Earth orbit, *even if (soft) commercial RAM was used*. This result points to the possibility of certifying before flight whether, for a given mission, commercial RAM provides sufficient reliability—a condition that we can evaluate empirically with BITFLIPS.

In future work, we aim to conduct the same evaluation using larger data sets, and to run tests with the higher radiation rates that would be experienced in Jupiter or Saturn orbit. BITFLIPS could also be applied to all flight software planned to be operated in a high-radiation environment, to assess risk quantitatively.

ACKNOWLEDGMENTS

Our experiments used the JPL Supercomputing Facility, which is funded by the JPL Office of the Chief Information Officer. We gratefully acknowledge the support of a JPL Lew Allen Excellence in Research award, which supported this research. This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

Alsabti, K., Ranka, S., and Singh, V. (1998). An efficient k-means clustering algorithm. In *Proceedings of the 1st Workshop on High Performance Data Mining*.

Asuncion, A. and Newman, D. (2007). UCI machine learning repository. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Bradley, P. S. and Fayyad, U. M. (1998). Refining initial points for k-means clustering. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 91–99.

Castano, R., Mazzoni, D., Tang, N., Doggett, T., Chien, S., Greeley, R., Cichy, B., and Davies, A. (2005). Learning classifiers for science event detection in remote sensing imagery. In *Proceedings of the Eighth International Symposium on Artificial Intelligence, Robotics, and Automation in Space*.

Castano, R., Wagstaff, K. L., Chien, S., Stough, T. M., and Tang, B. (2007). On-board analysis of uncalibrated data for a spacecraft at Mars. In *Proceedings of the Thirteenth International Conference on Knowledge Discovery and Data Mining*, pages 922–930.

Cortes, C. and Vapnik, V. (1995). Support-vector network. *Machine Learning*, 20:273–297.

Granat, R., Wagstaff, K. L., Bornstein, B., Tang, B., and Turmon, M. (2009). Simulating and detecting radiation-induced errors for onboard machine learning. In *Proceedings of the Third IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 125–131.

Huang, K.-H. and Abraham, J. A. (1984). Algorithm-based fault tolerance. *IEEE Transactions on Computers*, 33(6):518–528.

Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of Classification*, 2:193–218.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C., Silverman, R., and Wu, A. Y. (1999). Computing nearest neighbors for moving points and applications to clustering. In *Proceedings of the Tenth ACM-SIAM Symposium on Discrete Algorithms*, pages S931–S932.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, volume 1, pages 281–297.

Nethercote, N. and Seward, J. (2007). Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 89–100.

Pelleg, D. and Moore, A. (1999). Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the Fifth International Conference on Knowledge Discovery in Databases*, pages 277–281.

Shirvani, P. P., Saxena, N. R., and McCluskey, E. J. (2000). Software-implemented EDAC protection against SEUs. *IEEE Transactions on Reliability*, 49(3):273–284.

Turmon, M., Granat, R., Katz, D., and Lou, J. (2003). Tests and tolerances for high-performance software-implemented fault detection. *IEEE Transactions on Computers*, 52(5):579–591.

Wagstaff, K. L. and Bornstein, B. (2009). K-means in space: A radiation sensitivity evaluation. In *Proceedings of the Twenty-Sixth International Conference on Machine Learning*, pages 1097–1104.