Proactive Storage Health Management to Reduce Data Center Downtime

(General Machine Learning)

Guanghua Shu
gshu@stanford.edu

Ji Eun Jang
jiejang@stanford.edu

December 15, 2017

1 Abstract

The project adopts machine learning algorithms to perform proactive storage health management to reduce data center down time. Specifically, we develop reliable algorithms that can predict disk drive failures for timely replacement. The dataset is a collection of SMART disk drive attributes provided by Back-Blaze. To construct compact time-series for model development, three techniques are used to process SMART attribute data, including time-series averaging, feature selection, and class balancing. Models based on four algorithms, including logistic regression (LR), support vector machine (SVM), extreme gradient boosting (XGBoost), and recurrent neural network (RNN), are developed and evaluated using prediction accuracy and F-score along with precision and recall. The baseline model LR achieved 87.2% prediction accuracy and a F-score of 0.87 with relative low recall of 0.82 for failed disks. Both prediction accuracy and F-score are improved in SVM, XGBoost, and RNN. Specifically, XGBoost and RNN achieved 94.9% and 94.1%, respectively with F-scores above 94%. More importantly, recalls of failed disks are improved to above 93% to greatly reduce the risk of false negative prediction. Finally, proactive prediction capability is explored to provide replacement suggestion in advance. RNN is able to keep prediction accuracy above 70% with 9 days of proactive time.

2 Motivation

It is critical to keep data centers continuously on. The costs of data center downtime have increased from $5,617/minute in 2010 to $8,851/minute in 2016 according to a study conducted on 63 data centers in the U.S. [1]. IT equipment failure is the most costly reason for data center downtime and storage components have the most frequent failing rate in current IT environments. Therefore, accurate failure prediction and timely replacement can decrease downtime costs and improve system reliability [2]. The defense mechanisms such as RAID to improve disks availability and reliability are not enough to prevent the failures to severely impact data center operations [2]. There exists monitoring systems such as SMART (self-monitoring, analysis, and reporting technology) to continuously report drive health status [3]. However, SMART has a large number of attributes and their effects on disk failure are poorly defined. In this project, we explore both traditional machine learning algorithms and deep learning methods to predict disk failure based on time-series of historic disk status.

3 Related Work

Many of existing SMART-based monitoring systems are simple and threshold-based normalization, which shows weak prediction power with failed disk recall score of merely above 50% [2, 4]. There exist statistical approaches to improve prediction accuracy [5, 6]. Murray et al. implemented Naive Bayesian classifier and Support Vector Machine, but achieved less than 60% detection rate [7]. More recently, Botezatu et al. showed very high accuracy using regularized greedy forest (RGF), random forest (RF), and support vector machine (SVM) [2]. Our project adopts and expands some of data preprocessing techniques from [2], and applies different machine learning algorithms including XGBoost and RNN.

4 Processing SMART

4.1 What is SMART?

Self-monitoring, analysis, and reporting technology (SMART) is a monitoring system for hard disk drives (HDDs) and solid-state disk drives (SDDs). SMART collects and reports on various attributes related to drive reliability. There exist SMART datasets collected and published by Backblaze [3]. Among Backblaze datasets, we choose Seagate disk drive dataset from year 2013 to 2017 to develop and test the algorithms.

It is difficult to train models with raw SMART data set mainly due to three issues. First, time-series SMART data needs to be averaged to reduce noise from noisy measurement as well as disk recovery mechanisms [2]. Effective averaging window sizes for each attributes can be different as each attributes are correlated to different types of failure mechanisms. Second, feature selection is necessary to decide which attributes are most indicative of impending disk failure. SMART standard defines more than 90 attributes. Moreover, each disk model has different dominant failure mechanisms, and hence different sets of effective attributes. Third, as disk drives have a low failure rate, the data set is heavily class-imbalanced. Without proper class balacning, classificaion accuracy for the class with a small training size (i.e. failed disks) can be significantly degraded. The rest of this section addresses these three issues.

4.2 Window Averaging

An averaging window size is decided by change point analysis. Our change point analysis build a time series model based on linear local trend model in (4.1), where y_t is observation at time t, μ_t is a slowly varying component called trend, ν_t is a slope term generated by a random walk [8]. Three model parameters, σ_ϵ, σ_ξ, and σ_ζ are estimated by maximizing its log-likelihood [9].

$$\begin{aligned} y_t &= \mu_t + \epsilon_t & \epsilon_t &\sim \mathcal{N}(0, \sigma_\epsilon^2) \\ \mu_{t+1} &= \mu_t + \nu_t + \xi_t & \xi_t &\sim \mathcal{N}(0, \sigma_\xi^2) \\ \nu_{t+1} &= \nu_t + \zeta_t & \zeta_t &\sim \mathcal{N}(0, \sigma_\zeta^2) \end{aligned} \tag{4.1}$$

Assuming there exists a change point τ when a meaningful change occurs in attribute values, we build two linear local trend models before and after τ. Then, an optimal change point τ_{max} is decided to maximize sum of log-likelihood of two models. If parameters of two models before and after τ_{max} are significantly different, the attribute is indicative of disk failure and the measured change point is valid.

$$\tau_{max} = \underset{\tau}{\operatorname{argmax}} \log(p(y_{1:\tau-1}|\theta_1)) + \log(p(y_{\tau:t}|\theta_2)) \tag{4.2}$$

Table 1: Change point analysis results

ID	Mean of CPs	ID	Mean of CPs
5	12	187	17
190	14	193	17
194	17	197	17
198	17	240	22
241	20	242	15

Table 2: Accuracy with class balancing for linear SVM

Balancing technique	Accuracy
Random sampling	91.5%
K-means sampling	94.1%
Weighted SVM	89.0%

We use mean values of valid change points to adjust an effective window size of an averaging function. Table 1 lists mean values of valid change points for some attributes. Then, raw values of the attributes are averaged with exponentially decreasing weights as Equation (4.3), where the parameter α controls its smoothing effect. We relate the mean values of valid change points to α as Equation (4.3) [10].

$$s_t = \alpha \times y_t + (1 - \alpha) \times s_{t-1} \text{ with } \alpha = \frac{2}{\tau + 1} \tag{4.3}$$

4.3 Feature Selection

To decide which attribute is most indicative, we measured accuracies of support vector machine (SVM) when the model is trained with only one attribute. Seagate drive model ST4000DM000 in Backblaze dataset provides 24 attributes. Fig. 1 shows test accuracy for each attribute. Among all 24 attributes, the attribute 193 (load cycle count) was most effective showing 80% accuracy, followed by attribute 187, 194, and so on.

To measure accuracy improvement by adding features, we added features in an order of higher accuracy shown in Fig. 1. Fig. 2 plots test and training accuracies over the number of features. The test accuracy improved as more features are added and saturated with more than 15 features. We didn't observe overfitting up to 24 attributes.

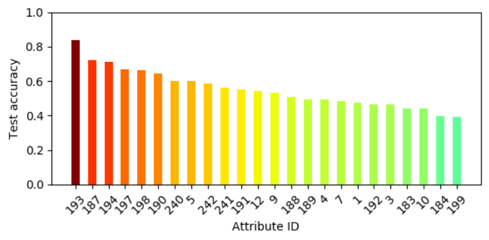

Figure 1: SVM test accuracy with only one attribute

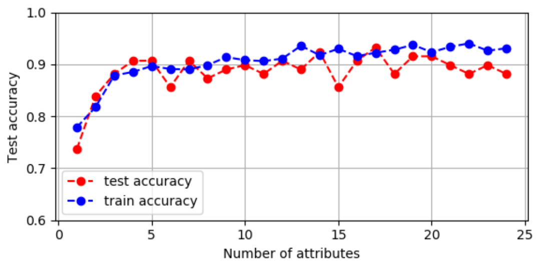

Figure 2: SVM test accuracy with multiple attributes

4.4 Class Balancing

To address the imbalanced-class issue, we tried three techniques: random sampling, K-means sampling, and weighted support vector machine. Random sampling blindly chooses the same number of healthy disks as the number of failed disks. K-means sampling divides the healthy disk data set to the same number of clusters as the failed disks and use cluster centroids as a representative healthy disk data set. Weighted support vector machine sets the ratio of penalties for different classes to the inverse ratio of the training class sizes [11].

Table 2 lists test accuracy with one of the three downsampling technique. Among three techniques, K-means sampling gave the best accuracy of 94.1%. However, applying K-means clustering on a healthy disk data set with a large number of samples took a significant computing time. Therefore, accuracy testing for the rest of the paper used random sampling.

5 Methods

Four algorithms are explored including logistic regression (LR), support vector machine (SVM), extreme gradient boosting (XGBoost), and recurrent neural network (RNN). A brief description of each algorithm is provided with the same settings for the dataset. In general, the disk failure detection is treated as a binary classification problem. The target labels can take only two values $y \in \{-1, +1\}$. Each input sample ($x \in \mathbf{R}^n$) is assumed to be independently and identically sampled from real world data distribution (iid) and m is the number of training samples.

5.1 Logistic Regression

The probabilistic view of LR is presented with the help of *sigmoid* function ($g(z)$) that converts the results of hypothesis into probability model. With the iid assumption for samples and labels $y \in \{-1, +1\}$, the log-likelihood can be derived as [12]

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^{m} \log p(Y = y^{(i)} | x^{(i)}; \theta) \\ &= -\sum_{i=1}^{m} \log\left(1 + \exp(-y^{(i)}\theta^T x^{(i)})\right) \qquad (5.1)\end{aligned}$$

Where parameter $\theta \in \mathbf{R}^n$ is to be learned during training and $y\theta^T x$ is known as margin. Intuitively, when margins are positive (predictions are correct and y has the same sign as $\theta^T x$), $\ell(\theta)$ approaches maximum value 0. Conversely, negative margins result in $\ell(\theta)$ stays more negative. The goal of LR algorithm is to maximize the log-likelihood in Equation (5.1). Finally, the LR algorithm has been implemented in Tensorflow [13] framework (LR from scikit-learn library[11] is used to confirm the correctness of our own implementation and for cross-validation). Same as for other algorithms described later, mini-batch Gradient Descent is used to learn the parameter θ either by maximizing likelihood or minimizing loss. The learning rate and batch size are the two hyper-parameters been tuned in the process.

5.2 Support Vector Machines

The goal of support vector machine is to find a hyperplane that separates the datasets with maximized minimum margins (distances from support vectors to hyperplane). In SVM, *hinge* loss is generally used in the loss function. Including the L2-regularization, the loss function for linear SVM becomes

$$J(W, b) = \frac{1}{m}\sum_{i=1}^{m}[\max(0, 1 - y^{(i)}(Wx^{(i)} + b)] + \frac{\lambda}{2}\|W\|^2 \qquad (5.2)$$

Both Linear SVM and SVM with Gaussian kernel (please refer to [12]) give similar performance in this dataset. Fro simplicity, only the linear SVM performance is reported in the results session.

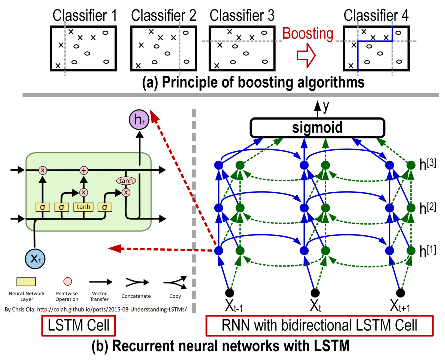

Figure 3: Summary of XGBoost and RNN algorithms.

5.3 Extreme Gradient Boosting

Extreme gradient boosting (XGBoost) [14] belongs to a family of boosting algorithms that convert weak learners into strong learners (classifiers in classification problems). A weak learner is assumed to be slightly better than random guessing. Fig. 3(a) illustrates the key principle of boosting process in general. Basically, each classifier can only classify the dataset correctly in partial, and the latter Classifier does better in the region that the previous Classifier does poorly. For example, Classifier 2 classifies the right side of the samples correctly which Classifier 1 is not able to do. Essentially, boosting is a sequential process that slowly learns from data and improves the prediction in subsequent iterations.

For XGBoost, each classifier in Fig. 3(a) is a decision tree $f_t(x)$, which is usually defined as

$$f_t(x) = w_{q(x)},\ w \in \mathbf{R}^T,\ q : \mathbf{R}^d \to \{1, 2, ..., T\} \quad (5.3)$$

Where w is the vector of scores on leaves of the trees, q is a function mapping each data point to the corresponding leaf, and T is the number of leaves. In XGBoost, the loss or objective function is [14]

$$L(f) = \gamma T + \lambda \|w\|^2 \quad (5.4)$$

The XGBoost algorithm used in this project is provided by xgboost package [15].

5.4 Recurrent Neural Network

Recurrent neural networks (RNN) provide a mechanism to handle datasets with temporal information (time series is a typical example). Although RNNs are believed to be able to handle long-term dependence, in practice, vanilla RNNs do not seem to be capable of learning them. The problem was explored in depth by Bengio, et al. [16]. The long short term memory (LSTM) cell, proposed by Hochreiter and Schmidhuber [17], is designed to remember information for long periods so as to alleviate the long-term dependence problem. One RNN example, built from bidirectional LSTM cell, is shown on right side of Fig. 3(b). Specifically, the solid arrows in blue represent one direction and the dash arrows in green stand for the revered direction. Fig. 3(b) also shows the details of a LSTM cell, adopted from Colah[18]. One LSTM cell has four neurons, interacting in a very special way. Three are with *sigmoid* activation and one uses *tanh* activation. The update of LSTM cell in each time step is

$$\begin{aligned}
&\text{Input gate: } i_t = \sigma(W^{(i)} x_t + U^{(i)} h_{t-1}) \\
&\text{Forget gate: } f_t = \sigma(W^{(f)} x_t + U^{(f)} h_{t-1}) \\
&\text{Output gate: } o_t = \sigma(W^{(o)} x_t + U^{(o)} h_{t-1}) \\
&\text{New memory cell: } \tilde{c}_t = \tanh(W^{(c)} x_t + U^{(c)} h_{t-1}) \\
&\text{Final memory cell: } c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
&\text{Final hidden state: } h_t = o_t \odot \tanh(c_t) \qquad (5.5)
\end{aligned}$$

Keras [19] is used to implement RNN using bidirectional LSTM cells with Tensorflow as backend to perform backward propagation in time for gradient calculation. Adam optimizer [20] is employed to update parameters in Equation (5.5) with learning rate decay. We also tunned hyper-parameters including learning rate, batch size, number of hidden layers, hidden layer dimension, and dropout rate.

6 Results and Discussion

Table 3 summarizes the performance of four algorithms discussed in previous section. The dataset has 4600 samples, in which 80% is training set, 10% is development set and the remaining 10% is test set. In addition to prediction accuracy, performance of all algorithms are also evaluated with precision (P), recall (R), and F-score (F) defined in Equation (6.1).

$$P = \frac{tp}{tp + fp} \quad R = \frac{tp}{tp + fn} \quad F = \frac{2PR}{P + R} \quad (6.1)$$

where $tp, fp,$ and fn are true positive, false positive, and false negative, respectively. Results of the four algorithms are summarized in Table 3. Comparing to results presented in the poster session, the performance improvement are due to two main factors:

Table 3: Algorithm performance summary

		LR	SVM	XGBoost	RNN
Accuracy [%]	Train	90.6	91.5	99.9	97.2
	Test	87.2	88.7	94.9	94.1
Failed	Precision	0.92	0.96	0.96	0.95
	Recall	0.82	0.89	0.93	0.93
	F-score	0.87	0.93	0.95	0.94
Healthy	Precision	0.83	0.90	0.93	0.94
	Recall	0.93	0.93	0.97	0.95
	F-score	0.88	0.97	0.95	0.94

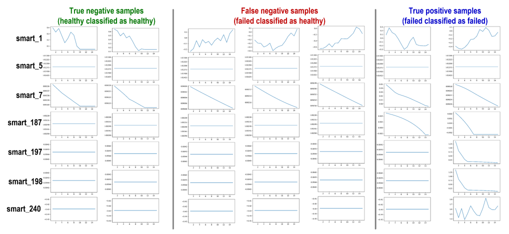

Figure 4: Error analysis for RNN model.

i) increase of dataset size (about 5x); and ii) accurate smoothing window for each feature (20 days was applied to all features in previous results).

6.1 Error Analysis

False negative error, predicting failed disk as healthy ones, is most costly since undetected disk failures directly lead to increase of data center down time. This is directly relate to recall of failed disks (also precision of healthy disks) in Table 3. Although precision and recall are fairly well balanced, there does exist a uniform trend of lower recall in failed disks and lower precision in healthy ones which suggests that false negative error is one of the main errors. Here, we want to look into the false negative cases in RNN model to gain some understanding about the reason why the model makes such errors. Two observations can be made on Fig. 4 showing true negative, false negative and true positive samples with 7 different features. First, comparing false negative to true negative samples, it is not surprising that false negative happens since most of the attributes are essentially the same. The only slight difference is that **smart_1** in false negative case tends to have more negative part and random fluctuation. Second, comparing false negative to true positive vs, this tends to suggest that in order for a sample to be classified as positive, in addition to the negativity and random fluctuation in **smart_1**, other attributes should also

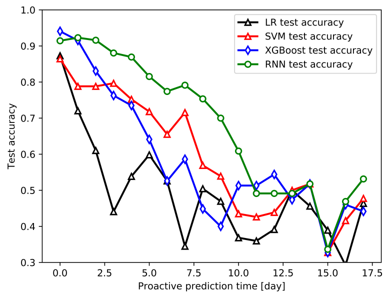

Figure 5: Proactive prediction accuracy.

have some kind of variations that related to failures.

6.2 Proactive Failure Prediction

Proactive prediction capability is also evaluated. With models only trained once on time series of complete disk life cycles, we test the accuracy of trained models with test time series that have last n days close to the end of the disk life cycles been removed (proactive time). Such accuracy is defined as proactive prediction accuracy. The experiment results in Fig. 5 show that proactive accuracy drops as proactive time increases for all algorithms, since the test samples bear less similarity to the training set with increasing proactive time. Note that the proactive prediction accuracy in RNN decreases slower compared to other models as the proactive time increases, suggesting RNN has better proactive prediction capability. This may be related to the inherent nature of RNN, designed to preserve long-term information.

7 Conclusion and Future Work

XGBoost achieved best prediction accuracy of 94.9% with RNN closely follows (94.1%) and F-scores and recalls for failed disks are high for both. This suggests both XGBoost and RNN can fit time series and generalize well enough on test set with complete disk life cycles. However, RNN keeps proactive prediction accuracy above 70% with proactive time up to 9 days, while the accuracy for XGBoost drops below 70% beyond 5 days. This suggests RNN may generalize better with long-term dependence.

Two future directions for this project: i) transfer learning between different disk models that have slightly different attributes; ii) develop algorithm for better proactive prediction accuracy.

8 Contributions

Both group member have been actively contributing to the projects. A lot of study and discussion have been helpful to improve the project. We have been working interactively, with Ji Eun leading data preprocessing and Guanghua focusing on building algorithms and evaluating the performance.

References

[1] P. Institute, "Cost of data center outages," 2016. [Online]. Available: https://www.ponemon.org/blog/2016-cost-of-data-center-outages

[2] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann, "Predicting disk replacement towards reliable data centers," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 39–48.

[3] B. Beach, "Hard drive smart stats," 2014. [Online]. Available: https://www.backblaze.com/blog/hard-drive-smart-stats/

[4] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population." in *FAST*, vol. 7, no. 1, 2007, pp. 17–23.

[5] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 350–357, 2002.

[6] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, no. 1, 2007, pp. 1–16.

[7] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *Journal of Machine Learning Research*, vol. 6, no. May, pp. 783–816, 2005.

[8] J. Commandeur and S. Koopman, , *An Introduction to State Space Time Series Analysis*. Oxford, 2007.

[9] J. Perktold, S. Seabold, and J. Taylor, "Statsmodels: statistics in python," Oct. 2017. [Online]. Available: http://www.statsmodels.org/stable/index.html

[10] "Exponentially weighted windows," Oct. 2017. [Online]. Available: http://pandas.pydata.org/pandas-docs/stable/computation.html#exponentially-weighted-windows

[11] 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.

[12] A. Ng, "Support vector machines," Dec. 2017. [Online]. Available: cs229.stanford.edu/notes/cs229-notes2.pdf

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[14] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[15] D. D. M. L. Community, "Scalable, protable and distributed gradient boosting," Aug. 2015. [Online]. Available: https://github.com/dmlc/xgboost

[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar 1994.

[17] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Advances in Neural Information Processing Systems 9*. MIT Press, 1997, pp. 473–479.

[18] C. Colah, "Understanding lstm networks," Aug. 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[19] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980