Classifying Server Behavior and Predicting Impact of Modernization Actions

Jasmina Bogojeska*, David Lanyi*, Ioana Giurgiu*, George Stark† and Dorothea Wiesmann*
*IBM Research - Zurich, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland, {jbo, dla, igi, dor}@zurich.ibm.com
†IBM Global Technology Services 294 Route 100 Somers, NY 10589, USA

***Abstract*—Today the decision of when to modernize which elements of the server HW/SW stack is often done manually based on simple business rules. In this paper we alleviate this problem by supporting the decision process with an automated approach based on incident tickets and server attributes data. As a first step we identify and rank servers with problematic behavior as candidates for modernization using a random forest classifier. Second, this predictive model is used to evaluate the impact of different modernization actions and suggest the most effective ones. We show that our chosen model yields high quality predictions and outperforms traditional linear regression models on a large set of real data.**

I. Introduction

The business climate in recent years has required IT service providers to continually lower the cost of services delivery while improving the delivery quality. Although IT service delivery quality has many dimensions, system availability is clearly the key metric. Consequently, because of its direct impact on system availability, Incident and Problem Management has received particular attention in the quest for ever higher quality and lower cost. According to the IT Infrastructure Library (ITIL), Incident Management is a discipline tasked with restoring normal operations to a service when it has been degraded or is down. Whereas the primary objectives of Problem Management are to prevent problems and the the resulting incidents from happening, to eliminate recurring incidents, and to minimize the impact of incidents that cannot be prevented [1]. However, recent innovations in Incident and Problem Management include the additional objectives of automatic classification of incident tickets for optimal routing to the best resolution teams, aggregation of relevant system information and similar incident tickets for faster root cause analysis and tickets resolution, as well as optimal staffing of resolver teams [2], [3], [4], [5]. While root cause analysis across similar incident tickets as part of a continuous improvement lifecycle [5], may lead to the identification of troublesome server configurations, none of the aforementioned approaches systematically investigates the impact of server attributes and their modernization on the level of incident tickets. In general, while the drivers for different server modernization actions are known, the process of deciding which action to apply at which point in time is often manual and following simple rules. For example, a number of reasons typically drive server hardware refresh: increased risk of malfunctioning or end of support of the old hardware as well as higher performance for increased workloads, enhanced support for virtualization, and lower power consumption of the new hardware [6]. Yet, while the drivers are principally understood, most IT departments still follow a fairly rigid strategy of replacing servers at three to five years of age. In this paper we introduce a novel, automated approach to selecting appropriate server modernization action based on actual server behavior. Our method is adopting the rationale underlying condition based maintenance and predictive maintenance in asset-intensive industries, namely to infer time and type of maintenance from the historic asset monitoring signals. The contributions of this paper are: (1) A Random Forest model that predicts from the server HW, OS, and utilization properties whether the number of incident tickets for this server exceeds a pre-defined threshold, thus classifying the server as *problematic*. (2) The application of the predictive model to evaluating the impact of modernization actions for the *problematic servers*. While our model does not include some of the more extrinsic modernization drivers, e.g. decreasing power consumption of new server HW, it represents a fundamental shift from rule-based maintenance towards a modernization strategy based on actual and improved system incidents after implementation of modernization actions. Our approach thus offers businesses and IT service managers the possibility of making smarter and more economical decisions. The remainder of the paper is organized as follows: In Section II we compare and contrast our approach with existing approaches to predictive maintenance and rejuvenation in related fields. Section III describes the predictive model candidates and the problem setting. The Experimental Section IV presents the performance of the chosen Random Forest Model and its usage to predict the impact of various modernization actions.

II. Related Work

This section positions our work in the context of IT service management. We identify three important directions that contribute towards improving different levels of the SW/HW stack: (1) incident and change request analysis, (2) predictive maintenance, and (3) software aging analysis.

Incident and Change Request Analysis. In problem management, the current trend is to move from reactive solutions to proactive strategies, where we seek to identify the cause of incidents and define corrective actions in advance. Diao et al. [2] propose a rule-base crowdsourcing classification method thus facilitating root cause analysis of incidents and failure trend monitoring. Compared to supervized learning methods, their approach achieves higher accuracy especially

ISBN 978-3-901882-53-1, 9th CNSM and Workshops ©2013 IFIP

when the cost of labeling incidents into failure classes is high. Gupta et al. [3] propose a set of techniques for multi-dimensional knowledge integration. Similar to [2], the authors investigate incident classification, but use machine learning to generate ticket failure classes. Further, they automatically associate resources to tickets and collect system vitals through monitoring tools, such as IBM Tivoli Monitoring [7]. The appropriate incident resolution remains to be determined by the system administrator based on the facts aggregated by the system. Lucca et al. [4] focus on clustering incidents into failure classes, in order to route the corresponding tickets to those teams that possess the proper skills to resolve them. All these techniques are orthogonal to our model, as they tackle the ticket classification aspect of the management process. However, we envision that future predictive analysis focused on specific ticket classes would benefit from such approaches.

Incident tickets can either be human generated or thrown by monitoring tools. While the first category always requires corrective measures, the same is not always true for the second one. In this context, Tang et al. [8] propose a technique to reduce the number of non-actionable tickets, while preserving all automated tickets that require such measures. Their evaluation shows an accuracy of correctly detecting non-actionable items of up to 75%. We recognize the benefits of using such a technique to eliminate non-actionable tickets for future predictive models. Orthogonal approaches that detect actionable patterns [9], [10] could be used in a complementary fashion to produce even better results.

Finally, previous research has shown that faulty changes can increase critical system and application outages. Kadar et al. [11] focus on applying automatic methods to classify change requests, such that one can have a better understanding of past failure reasons for similar requests. Similarly, Santos et al. [12] propose an interactive and dynamic method to quickly identify root causes for failures due to change requests. Motivated by the negative impact of such changes on the entire system, the proposed techniques can be used together with our predictive model to reduce the overall number of future incidents.

Software Aging and Rejuvenation. Software aging [13] refers to the progressive performance degradation or increased occurence of failures of a software system. This is mainly due to the exhaustion of operating system resources or error accumulation. In recent years, a prevention technique called *rejuvenation*, has been proposed in the software field. It involves stopping the running software, cleaning its state and restarting it. However, the question of when to apply such rejuvenation or modernization techniques remains. Clearly, periodic actions may not produce the best results since software and hardware do not age at a constant rate. Grottke et al. [14] and Trivedi et al. [15] propose time series and stochastic models, respectively, to estimate trends and detect seasonal patterns for aging software. They also show how, using seasonal variation, one can predict future resource usages and trigger corresponding rejuvenation measures. while we share the similar goal of reducing server degradation, in contrast to [14], [15], our improvement target is reduction of server incident tickets by applying more complex improvement actions.

Predictive Maintenance. Oil, logistic or utility companies base their businesses on physical assets. To ensure success, they need to be able to maintain their systems in working order, minimize outages, as well as reduce maintenance and operational costs. Predictive maintenance is a crucial step and focuses on analyzing data patterns for the underlying infrastructure to offer insights and early warnings with respect to emerging problems. In [16], the authors propose several methods to assess the physical assets condition: (1) association analysis between failures and configuration, model, age or usage parameters; (2) lifetime analysis, that based on historical data can project the failure probability in the future; (3) risk estimation for networked infrastructure; (4) replacement planning. Such analytics is useful for understanding server behavior and we propose a technique that tackles both failure associations and replacement strategies. Moreover, we envision enriching our predictive approach in the future with time series analysis, to model different levels of incidents (e.g., failures, outages, performance degradation) based on historical data. In this context, we could apply the method proposed by Liu et al. [17], which learns temporal graph structures to detect anomalies in oil-production monitoring systems.

III. Methods

In this section we describe the problem setting and provide a summary of two statistical learning methods we decided to use and compare in the paper.

A. Problem Setting

Assume we are given a set of servers along with their configuration information and their incident tickets collected over a certain period of time from several service delivery centers. We then define the notion of *problematic server* as a function of ticket volumes and ticket severities. High severity incident tickets indicate severe system failures thus posing a high outage risk. Excessive numbers of low severity incident tickets indicate high system administration workload due to less critical system malfunctioning. Our goal is then to automatically identify and rank the problematic servers based on combined server configuration information and incident ticket data. We achieve this by applying the statistical learning methods described in the text below on the server data set.

Formally, let S denote the p-dimensional space of all possible values for p considered server features. Each server is then represented by a vector $\mathbf{x} \in S$ used as an input for a predictive model M. Once trained on the available set of servers, for each $\mathbf{x}$, M associates a probability for it to be a *problematic server*, i.e. $M(\mathbf{x}) \in [0, 1]$. We can then use $M(\mathbf{x})$ to rank all servers and identify the problematic ones (those with $M(\mathbf{x}) > 0.5$). Moreover, we can also use the predictive model to evaluate the impact of different server modernization actions and to suggest the most effective ones. Let $a : S \times P_a \mapsto S$ denote an arbitrary *parameterized improvement action*, a function which associates an input

vector $\mathbf{x}$ of a server and an action parameter $p \in P_a$ with the vector of the modified server features $\tilde{\mathbf{x}} = a(\mathbf{x}, p)$, after such an improvement action has been performed. This means that $\tilde{\mathbf{x}}$ represents $\mathbf{x}$ with its features adjusted according to the action's effect. We also consider combined actions. Let $(a_1, P_{a_1}), \ldots, (a_n, P_{a_n})$ be improvement action definitions. The combined action $(a^*_{1\ldots n}, P^*_{1\ldots n})$ denotes the composite action function $a_1 \circ \ldots \circ a_n$. It is assumed that $a_1(a_2(x, p_2), p_1) = a_2(a_1(x, p_1), p_2)$ for any two actions. With a defined set of improvement actions $A = \{(a_1, P_{a_1}), \ldots, (a_n, P_{a_n})\}$, a chosen server $\mathbf{x}$, and the predictive model M, we can simulate how different actions with different parameters (or their composites) change the probability of a server being problematic. To achieve this, we calculate the prediction for the modified server using M, i.e. $\tilde{p} = M(\tilde{\mathbf{x}})$, $\tilde{\mathbf{x}} = a(\mathbf{x}, p_a)$. We can measure the *improvement* of a parameterized action (a, p_a) by taking the difference between the prediction for the server before ($\mathbf{x}$) and after ($\tilde{\mathbf{x}} = a(\mathbf{x}, p_a)$) the modification , i.e. $I_{(a,p_a)} = M(\mathbf{x}) - M(\tilde{\mathbf{x}})$. This enables us to choose actions that yield high improvements.

B. Linear Logistic Regression

Let $\mathbf{x} \in R^p$ denote a real valued random input vector and y denote a random output variable with joint input-output distribution $p(\mathbf{x}; y)$. Assuming the existence of K classes and multinomial distribution of the output y, logistic regression uses linear functions in the input $\mathbf{x}$ to model the log-odds of the posterior probabilities of the classes $\{P(y = k|\mathbf{x}), k = 1, \ldots, K\}$. $P(y = k|\mathbf{x})$ is the conditional probability of a sample $\mathbf{x}$ to belong to class k. In order to ensure that the posterior probabilities of the K classes sum to 1, the model is specified with $K - 1$ log-odds (also termed logit transformations):

$$\begin{aligned} log\frac{P(y=1|\mathbf{x})}{P(y=K|\mathbf{x})} &= \beta_{10} + \beta_1^T\mathbf{x} \\ log\frac{P(y=2|\mathbf{x})}{P(y=K|\mathbf{x})} &= \beta_{20} + \beta_2^T\mathbf{x} \\ &\vdots \\ log\frac{P(y=K-1|\mathbf{x})}{P(y=K|\mathbf{x})} &= \beta_{(K-1)0} + \beta_{K-1}^T\mathbf{x}. \end{aligned}$$

The posterior probabilities derived from these equations are given by:

$$\begin{aligned} P(y=k|\mathbf{x}) &= \frac{\exp(\beta_{k0} + \beta_k^T\mathbf{x})}{1 + \sum_{i=1}^{K-1}\exp(\beta_{i0} + \beta_i^T\mathbf{x})}, k \leq K-1 \\ P(y=K|\mathbf{x}) &= \frac{1}{1 + \sum_{i=1}^{K-1}\exp(\beta_{i0} + \beta_i^T\mathbf{x})}. \end{aligned}$$

For a given training data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ the logistic regression model is usually estimated by a maximum likelihood approach.

In this paper we are interested in the binary classification scenario ($K = 2$) with $y \in \{0, 1\}$. Then the solution of:

Optimization Problem 1: Over parameters β and λ, maximize:

$$\sum_{(\mathbf{x}_i, y_i) \in D} \{y_i\beta^T\mathbf{x}_i - \log(1 + \exp(\beta^T\mathbf{x}_i)))\} + \lambda J(\beta).$$

is a *maximum a posteriori* of the logistic regression model for binary classification with parameters β and regularizer $J(\beta)$ controlled by the regularization parameter λ. In practice, only a finite sample of the data is available. The regularizer is then used to counter overfitting and thus improve the generalization performance of the fitted model on unseen data ([18], [19]). In our work we apply *ridge logistic regression* and *lasso logistic regression* that use the square of the L_2-norm of the model parameters β denoted by $\|\beta\|^2$ and the L_1-norm of the model parameters β denoted by $|\beta|$ as regularizer functions $J(\beta)$, respectively.

Linear classification models such as linear logistic regression are very well studied and frequently applied because they provide interpretable classification rules. Their main disadvantage is that they fail to capture complex dependencies which appear in the data from many real-world applications. Therefore we also used the random forest model, a non-linear statistical learning method described below.

C. Random Forest

Random forest models [20] are ensembles of classification or regression trees. While trees are very attractive and widely used nonlinear models due to their interpretability, they exhibit high variance. The random forest model reduces the variance by averaging a collection of decorrelated trees which provides a performance comparable to support vector machines (SVMs) and boosting methods. Usually, B trees are fitted using CART (Classification and Regression Trees) [19]; each tree on one of B bootstrap samples drawn from the training data. The trees are fully grown by considering only a fixed-size, random subset of all features when choosing the best split variable for their terminal nodes. They are left unpruned. The prediction for a new sample is computed as a majority vote or as an average of the predictions of all trees in the collection for classification and regression, respectively. A summary of the procedure for training random forest models is given in Algorithm 1.

An error estimate based on *out-of-bag* (OOB) samples is computed during the fitting procedure of a random forest model: the random forest prediction for each training sample is computed by averaging the trees induced on bootstrap samples which did not contain the considered sample. The OOB error estimate is comparable to the one obtained by a time-consuming cross-validation and can be used for model selection. The OOB samples are also used to produce a variable-importance measure that quantifies the prediction ability of each feature. It is computed as the decrease in model accuracy on the OOB samples of each tree when the values of the feature of interest are permuted, averaged over all trees. When using non-linear models variable-importance measures are very valuable since they provide model interpretability. We

TABLE I: List of server configuration predictors used in the classification models.

Predictor	Description
Purpose	Grouping of the server purpose in five classes: App, Dev, Net, Infr, Sto
Server Family	Manufacturer and machine architecture (e.g. IBM p series, HP ProLiant, etc.)
Age	Elapsed time since server release in years
Virtualization level	Number of logical servers on the same physical server
OS Family	Type of OS (e.g. IBM AIX, Windows)
OS currency	Equidistant mapping of the versions for each OS type (excluding patch levels) to numbers in the interval $[0, 1]$, where 0 indicates no information available and 1 indicates the most recent version
CPU busy	Mean CPU utilization over monthly periods averaged over 12 months
CPU max	Peak CPU utilization over monthly periods averaged over 12 months
Memory busy	Mean memory utilization over monthly periods averaged over 12 months
Disk busy	Mean disk space utilization over monthly periods averaged over 12 months

Algorithm 1 Train Random Forest Model for Classification or Regression

Input: Training data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, number of trees B, size of subset of predictors m used for growing the trees.
For $b = 1$ to B:

1) Generate a bootstrap data set of size N by drawing samples randomly with replacement from the training data D.
2) Grow a full-size tree T_b using the bootstrapped data set by recursively repeating the two steps below for each terminal node of T_b:
   - Randomly select a subset of m predictors ($m \leq p$).
   - Choose the best predictor and split point among them and split the node into two child nodes.

Output: Ensemble of trees $T_1, \ldots, T_B$.

should also pint out that in the classification scenario the OOB votes from the trees are used to compute class probabilities for the samples.

IV. Experiments and Discussion

A. Data

The data set used in our work is gathered from several accounts of a large IT service provider over a period of one year. It contains 10101 servers and 118121 incident tickets. As depicted in Figure 1 the different accounts are not represented with an equal number of training samples, i.e. number of logical servers. We extracted the following information for each logical server: account ID, server-hardware information, OS information, server purpose, and utilization information. A detailed list of the extracted predictors is given in Table I.

We assign labels to the servers according to the following definition. A server is *problematic* if it generates at least two tickets with high severity or at least twelve tickets with low severity within a year. Otherwise the server is *unproblematic*.

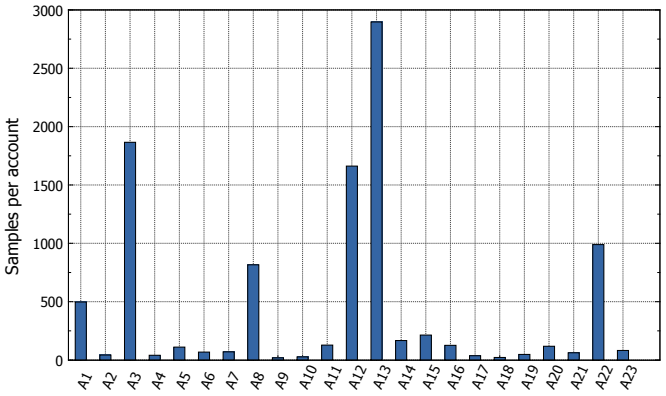


Fig. 1: Number of samples per account in the available data set.

B. Performance Measures

To evaluate the performance of the considered models we use two measures, namely classification accuracy and area under the receiver operating characteristic (ROC) curve (AUC). A ROC curve is a plot of the false positive rate (the fraction of false positives out of all negatives) against the true positive rate (the fraction of true positives out of all positives) for all possible decision thresholds. In this way it illustrates the ranking ability of a binary classification method. The area under this curve has values in $[0, 1]$, where 1 corresponds to perfect predictions and 0.5 to random guessing.

It is important to note that in the case of large class imbalance in the training data, the accuracy is not a proper performance measure, as the simple method that just assigns the majority class to every sample will achieve very high accuracy. Thus, when we quantify the performance of methods on class-imbalanced training data instead of accuracy and AUC, we report balanced accuracy, G-mean and F-score defined as:

$$
\begin{aligned}
Acc^{-} &= \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}} \\
Acc^{+} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \\
\text{Balanced Accuracy} &= \frac{Acc^{-} + Acc^{+}}{2} \\
\text{G-mean} &= \sqrt{Acc^{-} \cdot Acc^{+}} \\
\text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\
\text{Recall} &= Acc^{+} \\
\text{F-measure} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
\end{aligned}
$$

C. Results

This section describes the evaluation setting and presents the results of the computational experiments. We use a 10-fold cross validation procedure to select the regularization parameter λ in the two logistic regression models. The two model parameters of the random forest, i.e. the number of trees B and the number of variables considered at each split m are selected based on the OOB error. We report average accuracy and average AUC obtained from 100 runs, where in each run we randomly select 70% of all available data for training, fit the model of interest on them and compute the performance measure on the remaining (unseen) 30% of the data. The results for the three considered models, the ridge logistic regression, the lasso logistic regression and the random forest classifier are summarized in Figure 2.

Compared to the linear logistic regression models, the random forest model achieves both better accuracy and AUC performance. The improvement is 11% and 9.4% over ridge regression and lasso regression, respectively. According to Wilcoxon rank sum test the two improvements are significant with p-value $< 2.2e-16$. This demonstrates that a non-linear approach such as the random forest is better suited for tackling the server classification problem. Therefore, in the remainder of the paper we focus on the random forest method.

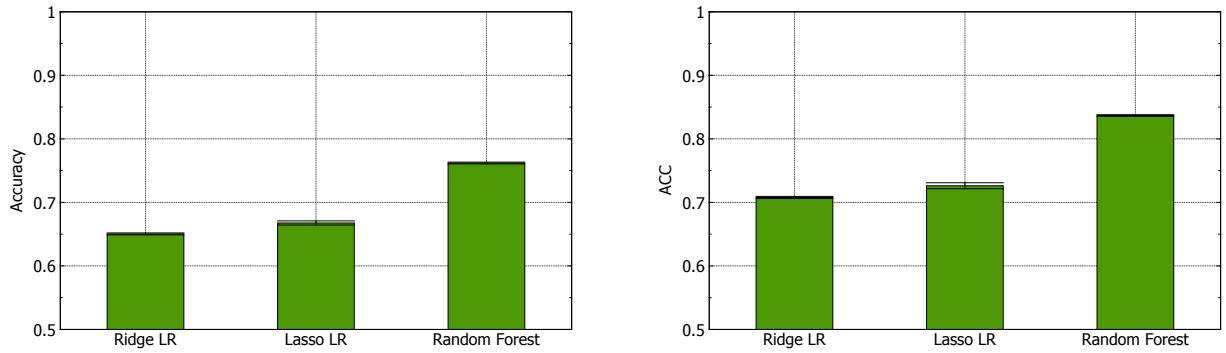
Similar to many other classifiers, the prediction performance of the random forest model is hampered by the problem of imbalanced data where at least one class is represented by a very small portion of the available samples. Since the random forest aims to minimize the overall error rate, it will achieve good prediction accuracy on the majority and poor accuracy on the minority class.

For the server classification problem it is important that we handle user-specified definitions for determining problematic servers. Therefore, it is often the case that in practice we need to deal with class-imbalanced data sets. To do this we adopt two approaches, namely, *balanced random forests* and *account-specific balanced random forests*. The balanced random forest algorithm [21] modifies step 1 of the original training procedure in Algorithm 1, such that first a bootstrap sample is drawn from the minority class and then a bootstrap sample with the same size is drawn from the majority class. In this way, on the one hand, each tree is trained on a class-balanced bootstrap sample, and on the other hand, since the majority class is resampled many times all available information is used by the model. Motivated by the fact that we have uneven sample representation for the different accounts, we devised the account-specific balanced random forest approach as follows. It creates balanced bootstrap training sets in Algorithm 1 by reducing the number of majority-class samples drawn from the over-represented accounts. The size of the reduction is proportional to the account size, i.e. the larger the account the smaller number of majority class samples are drawn from it. In this way we make sure that all available information from the accounts with fewer samples in the training set, is used.

In order to evaluate the performance of the two approaches described above, we modify the definition of problematic servers given in the Problem Setting section. In the new context, a server that generated more than 14 incident tickets across all severities in a year is a problematic server. This definition yields a class-imbalanced training set comprising 20% problematic and 80% unproblematic servers. In order to evaluate the performance of the two approaches that deal with imbalanced data sets (the balanced random forest and the account-specific balanced random forest), we use the same evaluation setting as the one described in the beginning of this section. The only difference is that we evaluate the models by using the performance measures for imbalanced data sets instead of using accuracy and AUC. Note that the held-out test sets in the evaluation setting are randomly sampled from the imbalanced training data set. The results of the computational experiments are shown in Table II.

When the training data are imbalanced, both the balanced random forest and the account-specific balanced random forest approaches significantly outperform the classical random forest method by $9 - 39\%$ for all considered performance measures with p-values$< 2.2e-16$ according to Wilcoxon rank sum test. Considering the performance of the two methods used for dealing with the class-imbalance problem, the balanced random forest achieves slightly better (around 2%) balanced accuracy and G-mean performance, and the account-specific balanced random forest approach yields 6% better F-score performance. According to Wilcoxon rank sum test all these differences are significant at the 0.001 level. In other words, while the balanced random forest approach achieves similar accuracy performance for both the majority and minority class, the account-specific balanced random forest approach provides better accuracy for the minority class by sacrificing little of the accuracy for the majority class.

To sum up, the two approaches we use in the case of imbalanced training sets achieve comparable balanced accuracy performance to the accuracy obtained for a balanced set.



(a) (b)

Fig. 2: Accuracies (a) and AUCs (b) with their corresponding confidence intervals for ridge logistic regression, lasso logistic regression and random forest. The depicted values are computed by averaging the accuracy and AUC obtained on a held-out data set over 100 runs.

TABLE II: Balanced Accuracy, G-mean and F-score with their corresponding standard errors (SE) for the classical random forest, the balanced random forest (BRF) and the account-specific balanced random forest approach.

Method	Balanced Acc (*SE*)	G-mean (*SE*)	F-score (*SE*)
Random Forest	0.651(0.001)	0.578(0.002)	0.498(0.003)
Balanced RF (BRF)	0.759(0.001)	0.758(0.001)	0.806(0.001)
Account-specific BRF	0.740(0.001)	0.737(0.001)	0.867(0.001)

TABLE III: Summary of considered modernization actions.

Action	**Description**	**Server Features Update**
OS refresh	Upgrade the OS to the latest version in its family	OS currency $= 1$
HW refresh	Change the underlying server hardware	Age $= 0$
Disk capacity increase	Increase the available disk space x times	Disk busy $=$ Disk busy/x
Virtualization reduction	Reduce the number of concurrently running virtual machines on one physical machine by $p\%$	Virtualization level $=$ Virtualization level$/p$ CPU busy (max) $=$ CPU busy (max) $/p$ Memory busy $=$ Memory busy $/p$

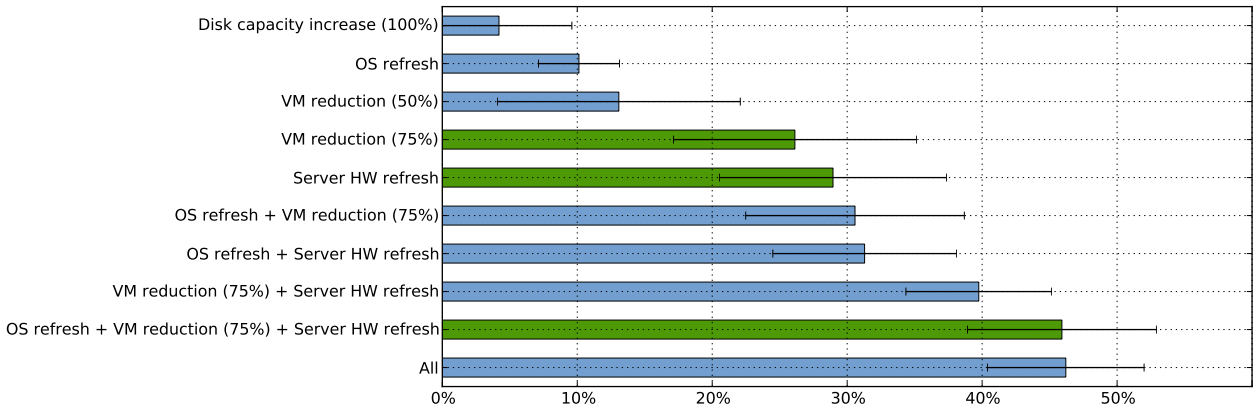


Fig. 3: Forecasted improvements after action implementation for a group of 7-year old OS Family 1 servers running on Server Family 1 architecture.

D. Use case: Impact of Server Modernization Actions

Server modernization leads to improved server behavior, such that the number and possibly even the severity of reported incidents is reduced. Applying such actions has been widely recognized as an important step in services management and predictive maintenance. To meet this end, we apply our

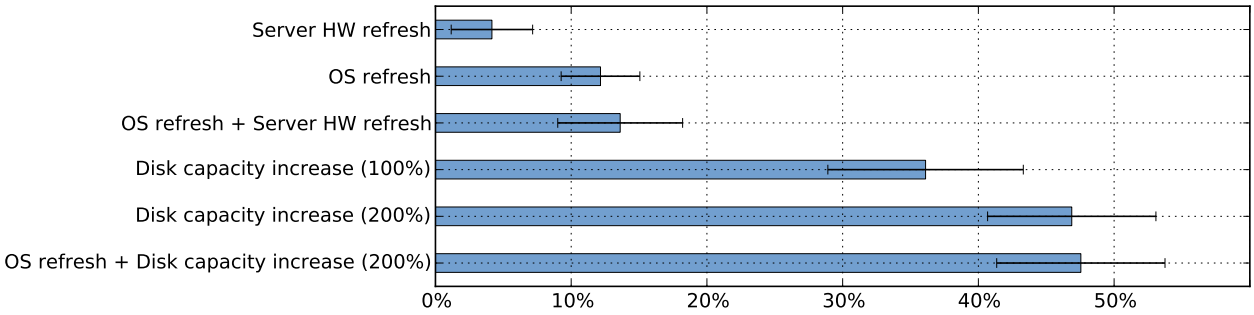


Fig. 4: Forecasted improvements after action implementation for a group of 7-year old OS Family 2 servers running on Server Family 2 architecture.

predictive model to evaluate the impact of single or combined modernization actions performed on servers or groups of servers as described in the Problem Setting.

As discussed in [22], several strategies can be applied in practice: (1) *centralization* – servers are consolidated in fewer sites, thus increasing availability and improving recovery capabilities; (2) *physical consolidation* – older servers are replaced with newer, more powerful or clustered systems; (3) *workload/application integration* – multiple applications are co-located onto fewer servers and OS instances; (4) *virtualization*, thus reducing physical complexity and increasing flexibility in resource allocation.

In our scenarios, we apply modernization actions at the level of small groups of servers, therefore centralization as a strategy cannot be applied. Similarly, since our data does not include information about the running workloads and their communication patterns, we exclude the third strategy as well. The improvement actions along with their description and translation in terms of server feature modifications are summarized in Table III.

We have chosen two groups of highly problematic servers (i.e. with the associated probability above 90%) with different sets of features in order to show how our approach can be applied in practice. The two groups are summarized in Table IV. The first group is used for application purposes and is therefore highly virtualized. On the other hand, the servers in the second group are dedicated storage machines and have high disk and memory utilization. In what follows we apply the model on a set of single and composed improvement actions, as defined above, and show which modernization strategies are most suitable for each of the chosen server groups. The results are summarized in Figures 3 and 4.

As observed in Figure 3 the servers in the first group would benefit most from single actions that either reduce the number of virtual machines or refresh the HW stack. This can be expected as the servers are 7 years old and highly virtualized. However, the highest improvement is achieved when considering both actions together combined with an OS refresh which is usually done when refreshing the hardware.

Considering that the second group consists of dedicated storage machines, the action with the highest impact increases the disk capacity as one would expect. These servers have

TABLE IV: Summary of the two server groups considered for modernization actions.

Property	**Group 1**	**Group 2**
Purpose	Application	Storage
Server Family	Server Family 1	Server Family 2
Age	7	7
OS Family	OS Family 1	OS Family 2
CPU busy	5.5	11.9
CPU max	61.5	98.4
Memory busy	43.3	90.7
Disk busy	45.1	75.6
Virtualization level	14.5	1

no virtualization consequently reducing the virtualization level cannot be applied in this scenario.

The results obtained from applying our predictive model on server groups with different properties lead to two important conclusions. First, there is no universal modernization strategy that provides the highest improvements in all cases. In fact, our evaluation shows that several factors, such as the server purpose, age or OS version, greatly influence the best suggested action. Second, contrary to the general belief that the most complex action (i.e. consisting of all single actions) should always be the best choice, in practice this is not necessarily the case. For instance, combining OS refresh and disk capacity increase actions for the second group of servers will provide minimal gains, but higher costs, when compared to applying only a disk increase strategy.

V. Conclusion

In this paper, we have introduced a novel, automated approach to selecting appropriate server modernization action based on actual server behavior. The core of the approach is a Random Forest model that predicts from the server HW, OS, and utilization properties whether the number of incident tickets for this server exceeds a pre-defined threshold, thus classifying the server as *problematic*. We have shown that the Random Forest model achieves an accuracy of 76% for balanced classes and outperforms the linear logistic regression

models on average by 10%. We further demonstrated the usage of the model to the evaluation of different server modernization actions for two groups of *problematic servers*. The highest-impact actions are a reduction of the number of virtual machines or refresh of the HW stack and a disk capacity increase for the first and the second server group, respectively. The results of the impact evaluation illustrate that no action is suited for all problematic servers and that more complex actions do not necessarily yield significantly higher gain. Including our predictive model in the server modernization decision process thus helps to identify the optimal modernization action.

ACKNOWLEDGMENT

The authors would like to express their gratitude to Nadeem Malik, Jorge Cordero, E.E. Jan, Yixin Diao, and Doug Dykeman, all employed by IBM, for helpful and constructive discussions that helped us improve the quality of the model.

REFERENCES

[1] http://www.itil officialsite.com/Publications/Core.aspx, "ITIL Service Operation."
[2] Y. Diao, H. Jamjoom, and D. Loewenstern, "Rule-based problem classification in it service management," *Proc. of IEEE CLOUD*, pp. 221–228, 2009.
[3] R. Gupta, H. Prasad, L. Luan, D. Rosu, and C. Ward, "Multi-dimensional knowledge integration for efficient incident management in a services cloud," *Proc. of IEEE SCC*, pp. 57–64, 2009.
[4] G. A. D. Lucca, M. D. Penta, and S. Gradara, "An approach to classify software maintenance requests," *Proc. of IEEE ICSM*, 2002.
[5] Y. Diao, A. Heching, D. Northcutt, and G. Stark, "Modeling a complex global service delivery system," *Proceedings of the 2011 Winter Simulation Conference*, pp. 690–702, 2011.
[6] J. S. Bozman and K. Broderick, "Server refresh: Meeting the changing needs of enterprise it with hardware/software optimization," *IDC Whitepaper*, 2010.
[7] http://www.ibm.com/software/tivoli/products/monitor/, "IBM Tivoli Monitoring."
[8] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik, "Optimizing system monitoring configurations for non-actionable alerts," *Proc. of IFIP NOMS*, pp. 34–42, 2012.
[9] J. L. Hellerstein, S. Ma, and C.-S. Perng, "Discovering actionable patterns in event data," *IBM Systems Journal*, vol. 43(3), pp. 475–493, 2002.
[10] W. Peng, C. Perng, and H. Wang, "Event summarization for system management," *Proc. of ACM KDD*, vol. 43(3), pp. 1028–1032, 2007.
[11] C. Kadar, D. Wiesmann, J. Iria, D. Husemann, and M. Lucic, "Automatic classification of change requests for improved it service quality," *Proc. of SRII Global Conference*, pp. 430–439, 2011.
[12] R. L. dos Santos, J. A. Wickboldt, R. C. Lunardi, B. L. Dalamzo, L. Z. Granville, L. P. Gaspary, C. Bartolini, and M. Hickey, "A solution for identifying the root cause of problems in it change management," *Proc. of IEEE/IFIP IM*, 2011.
[13] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: analysis, module and applications," *Proc. of 25th Symposium on Fault Tolerant Computer Systems*, pp. 381–390, 1995.
[14] M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi, "Analysis of software aging in a web server," *IEEE Transactions on Reliability*, 2006.
[15] K. S. Trivedi, K. Vaidyanathan, and K. Goseva-Popstojanova, "Modeling and analysis of software aging and rejuvenation," *Proc. of IEEE SS*, 2000.
[16] A. H. et al., "Analytics-driven asset management," *IBM Journal of Research and Development*, vol. 55(1-2), pp. 138–156, 2011.
[17] Y. Lui, J. Kalagnanam, and O. Johnsen, "Learning dynamic temporal graphs for oil-production equipment monitoring system," *Proc. of ACM KDD*, pp. 1225–1234, 2009.
[18] B. Schölkopf and A. Smola, *Learning with kernels*. MIT Press, 2002.
[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2009.
[20] L. Breiman, "Random forests," *Machine Learning*, 2001.
[21] C. Chen, A. Liaw, and L. Breiman, "Using random forest to learn imbalanced data," *Statistics Technical Reports, University of California Berkeley*, 2004.
[22] M. Badaloo, "An examination of server consolidation: trends that can drive efficiences and help businesses gain a competitive edge," *IBM Global Services*, 2008.