A Comparison of Machine Learning Algorithms for Proactive Hard Disk Drive Failure Detection

Teerat Pitakrat
University of Kaiserslautern
AG AQUA
Kaiserslautern, Germany
pitakrat@informatik.uni-kl.de

André van Hoorn, Lars Grunske
University of Stuttgart
Institute of Software Technology
Stuttgart, Germany
{van.hoorn,grunske}@informatik.uni-stuttgart.de

ABSTRACT

Failures or unexpected events are inevitable in critical and complex systems. Proactive failure detection is an approach that aims to detect such events in advance so that preventative or recovery measures can be planned, thus improving system availability. Machine learning techniques have been successfully applied to learn patterns from available datasets and to classify or predict to which class a new instance of data belongs. In this paper, we evaluate and compare the performance of 21 machine learning algorithms by using them for proactive hard disk drive failure detection. For this comparison, we use WEKA as an experimentation platform and benchmark publicly available datasets of hard disk drives that are used to predict imminent failures before the actual failures occur. The results show that different algorithms are suitable for different applications based on the desired prediction quality and the tolerated training and prediction time.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, availability, and serviceability*; I.5.4 [**Computing Methodologies**]: Pattern Recognition—*Applications*

Keywords

Machine learning; Proactive failure detection; Hard disk drive failures

1. INTRODUCTION

Reliability is one of the most important factors of critical systems to maintain its functionalities and provide services without disruption. In complex systems, most components are communicating with each other and a failure of one component may lead to a failure of another component. If the problem of a component persists and cannot be resolved, it might propagate to other parts of the system and cause a total failure.

The traditional approach is to prevent system failures in a reactive manner: when an internal misbehavior is detected, a monitoring agent triggers a recovery procedure—to avoid or alleviate the problem—and a human operator maybe informed. This method, however, is performed *after* a misbehavior has occurred, which may require some additional time until it is detected. This implies that when the recovery procedure starts, the problem may already have caused some damage to the system.

Proactive failure detection [39], on the other hand, aims to foresee an imminent problem by detecting early signs instead of detecting the problem itself. These signs include unusual behaviors of system parameters, such as, system load, CPU utilization, memory usage, network traffic, and hardware temperature. When a failure can be detected in advance, one or even more recovery actions can be carefully planned, analyzed, and evaluated to find the best solution for failure prevention [35]. Furthermore, in an extreme case when a failure cannot be avoided, other solutions, e.g. warming up spare components, can be initiated earlier to prepare for the service outage.

Machine learning techniques have been used in many studies (see, e.g., [3, 4, 31, 40]) to analyze the signs of a failure and to make a prediction whether the system is likely to fail in the near future. In order to apply machine learning techniques, the prediction problem is formulated into a binary classification problem where the signs obtained from system monitoring are learned and used as reference models to classify new runtime observations and concludes whether the system is about to fail.

In this paper, we compare the performance of machine learning techniques in terms of prediction quality, as well as time needed for training the models and for making predictions by applying it to a concrete dataset. We evaluate 21 compatible machine learning algorithms supported by WEKA [21], an open-source data mining tool, for the task of proactive failure detection in hard disk drives and make suggestions about which algorithm is most suitable under specific constraints.

The problem of the hard disk failure prediction was proposed by Hamerly and Elkan [22], who used supervised naïve Bayes and mixture models of unsupervised naïve Bayes trained with the expectation-maximization algorithm to predict failures. Hughes *et al.* [26] applied a rank sum test, which is a statistical hypothesis test to predict which drives will fail. The null hypothesis of the test is constructed by using the data from good drives. At runtime, when the parameter of the tested drive deviates significantly from the null hypothesis, a failure warning is issued. The dataset used in this paper was originally used by Murray *et al.* [36] to compare a newly developed classifier called the multi-instance naïve Bayesian classifier with a set of traditional algorithms, including support vector machines, unsupervised clustering, the rank sum test, and the reverse-arrangement test. The data set is thus also suitable to benchmark different machine learning algorithms in this study.

The remainder of this paper is organized as follows: Section 2 describes related evaluations of machine learning techniques in different areas. The machine learning algorithms that are compared in our experiments are explained in Section 3. Section 4 includes the description of our experiment and a discussion of results. Finally, conclusions are drawn in Section 5.

2. RELATED WORK

Our work aims to compare machine leaning algorithms for proactive detection of hard disk failures. In this section we review related work in the areas of general classification problems, computer science, and biomedical engineering.

Caruana and Niculescu-Mizil [9] evaluated ten supervised learning algorithms on eleven datasets of conventional binary classification problems. Eight performance metrics were used for the comparison of techniques. The result indicates that calibrated boosted trees performed best, followed by random forests and uncalibrated bagged trees. Lim *et al.* [32] performed an extensive comparison of 33 classification algorithms on 32 datasets. The training time of the algorithms and their prediction quality in terms of error rate are used as evaluation metrics (detailed in Section 4.3). They concluded that some algorithms perform slightly better than the others, but in practice, these numbers can be statistically insignificant. Therefore, when choosing among algorithms with similar performance, other factors such as training time should also be considered.

Mair *et al.* [34] compared machine learning algorithms, which are neural networks, case-based reasoning, and rule induction for predicting software development effort. The dataset was taken from 81 software projects each of which included ten project features, such as the number of entities and the team's and project manager's experience in years. The result shows that neural networks provide the best quality but that they are less configurable, which tends to decrease its practical usability. Abu-Nimeh *et al.* [1] compared six learning algorithms to classify legitimate and phishing emails. From the evaluation result, random forest appears to have the highest prediction quality but also with a high false positive rate. However, when the cost-sensitive measure is incorporated into the learning process, logistic regression achieves the best result with the lowest weighed error rate. Williams *et al.* [44] conducted a comparison of five algorithms to classify IP traffic flow in the network. The traffic properties, for instance protocol, packet length, and inter-arrival time, are extracted and used as features for the learning algorithms. They compared not only the classification quality but also the training and prediction time of each algorithm. The presented result concludes that even though C4.5 is slower in the model training, it is the fastest algorithm in making predictions at runtime.

In biomedical engineering, Chan *et al.* [10] compared six machine learning techniques to detect eye diseases. The classification quality are compared in terms of true positive rate, false positive rate, and area under ROC curve (AUC). The result shows that a mixture of Gaussian performs best, while quadratic discriminant analysis has a slightly lower prediction quality. Dreiseitl *et al.* [14] compared five techniques to classify three types of pigmented skin lesions. Their ROC analysis demonstrates that logistic regression, neural networks, and support vector machine outperform other techniques. Weiss and Kapouleas [43] conducted an empirical comparison of nine machine learning techniques which includes pattern recognition and neural networks. Four medical datasets were used for the evaluation, and the error rates are compared between each technique. They analyzed and discussed which technique configuration was best suited to a particular dataset.

3. MACHINE LEARNING ALGORITHMS

Machine learning algorithms have been widely used in various fields of research and have shown good performance in learning and recognizing patterns. In proactive failure detection, we make an assumption that before a failure occurs, there are parameters of the system that show early signs of the upcoming failure. When these data are collected and analyzed by the learning algorithms, certain characteristics of the system in good and faulty states can be modeled during the training phase and recognized at runtime.

This section gives a brief description of the 21 machine learning algorithms used in this paper. The selection criteria were the availability of an implementation in the WEKA tool and the compatibility with our dataset. With respect to the latter, note that, for example, M5P decision tree and simple linear regression cannot handle binary classification problems. We have divided their description into six categories following in Sections 3.1–3.6. Some section captions include the abbreviations used to refer to the algorithms in the remainder of this paper (see also Table 3).

3.1 Probabilistic Models

Probabilistic models use probability distribution of attributes in the dataset to build the models. At runtime, the probability of an instance belonging to each class is calculated.

3.1.1 Naïve Bayes Classifier (NBC)

Naïve Bayes classifier is a simple but powerful learning algorithm based on Bayes' theorem [33]. The learning phase analyzes the dataset and builds probability distribution models of the attributes. When the models are obtained, the prediction is carried out by calculating the probability of all attributes under the assumption that all attributes are independent and identically distributed.

3.1.2 Multinomial Naïve Bayes Classifier (MNB)

Multinomial naïve Bayes classifier is another application of the Bayes' theorem, which uses multinomial distributions as the underlying model instead of normal distributions. This technique allows the count of the occurrences for each value to be integrated into the model and has been successfully used in areas such as text classification (e.g., [27]).

3.1.3 Bayesian Network (BN)

Bayesian network is a directed acyclic graph that represents the conditional probability between each attribute [20]. The construction of the network comprises of two steps: building the structure of the network and estimating the probabilities. During runtime, the joint probability of an instance belonging to a class is calculated.

3.2 Decision trees

Decision trees are tree-like graphs, where each node contains a conditional statement that further splits the node into branches.

3.2.1 C4.5

C4.5 is a top-down decision tree, introduced by Quinlan [38], that employs a greedy algorithm to find the most important attribute at each step. At each level, the node is split until a leaf containing only instances from one class is achieved.

3.2.2 REPTree

REPTree constructs a decision tree by considering the information gain of all attributes in the dataset and splitting a node into further nodes. After the tree is built, the algorithm reduces the overfitting problem by using reduced-error pruning.

3.2.3 Random Forest (RF)

Random forest is a collection of decision trees proposed by Breiman [8]. This technique can be viewed as meta-learning [42], which improves the prediction quality by casting votes among the trees and assigning the most voted class to the predicted instance.

3.3 Rule-based Algorithms

Rule-based algorithms are based on a rule or a set of rules constructed by analyzing some characteristic or statistics of the training data.

3.3.1 ZeroR

ZeroR is the simplest classifier that is used to estimate the baseline for machine learning algorithms. The classification is done by classifying all instances as the majority class of the training set. For instance, the hard disk drive dataset used in our experiment contains 9,164 failing instances and 51,350 non-failing instances. Therefore, all instances during the prediction will be classified as non-failing.

3.3.2 OneR

OneR denotes a one-rule algorithm introduced by Holte [24] that employs only one rule to classify instances. This rule is constructed by building one rule for each attribute of the dataset and comparing their error rates. The rule of the attribute with the lowest error rate is chosen as the final rule and used in the classification.

3.3.3 Decision Table (DT)

Decision table, proposed by Kohavi [28], is a table containing a list of training instances with selected attributes. During the classification, an instance is compared to those in the list. If there is a match, it returns the majority class of those matched; otherwise, it returns the majority class of the whole list.

3.3.4 RIPPER

Repeated Incremental Pruning to Produce Error Reduction, or RIPPER, proposed by Cohen [12], builds rules by starting from an empty rule set and adding more rules until all positive instances are added.

3.3.5 PART

PART is a learning algorithm introduced by Frank and Witten [17]. The algorithm employs a divide-and-conquer approach to build the rule set. In each step, a partial decision tree is constructed and a rule is derived from the leaf of the tree that has the highest coverage. The whole process is repeated until the rule set covers all training instances.

3.4 Hyperplane Separation

Hyperplane separation employs a hyperplane in the multi-dimensional space of the dataset to separate all instances into classes.

3.4.1 Support Vector Machine (SVM)

Support vector machine, introduced by Cortes and Vapnik [13], is a technique that separates instances into two distinct classes by drawing a hyperplane between them. When working with a multiple-class classification problem, SVM classifies instances into one of the two main classes and further splits each class into smaller ones until the final class is obtained.

3.4.2 Sequential Minimal Optimization (SMO)

Sequential minimal optimization, proposed by Platt [37], is an improvement technique for SVM to speed up the training phase. SMO reduces the internal computation of quadratic problems into smaller sub-problems that can be solved analytically.

3.4.3 Stochastic Gradient Descent (SGD)

Stochastic gradient descent [6] is a stochastic optimization algorithm used to solve linear problems. In our experiment, the algorithm is used to build a linear model for SVM during the training phase.

3.5 Function Approximation

Function approximation estimates functions that map input vectors, which are extracted from an instance, to a value that represents an output class.

3.5.1 Simple Logistic Regression (SL)

Logistic regression [29] is a linear regression technique that can be used to predict binary-class instances. The algorithm uses LogitBoost [19] to build the regression model and was further improved by Sumner *et al.* [41] to increase the speed of the model construction.

3.5.2 Logistic Regression (LOG)

Logistic regression is similar to simple logistic regression but employs ridge estimators proposed by le Cessie and van Houwelingen [30] to reduce the error made by parameter estimation.

3.5.3 Multilayer Perceptron (MP)

Multilayer perceptron [23] is a type of neural network that contains multiple layers of neurons. Each neuron holds a function that maps an input variable to an output variable.

Serial no.	Hours before failure	Temp1	FlyHeight1	Servo8	ReadError17	WriteError	···
100001	2.216	10	7962	0	0	57005	···
100001	2.200	12	7972	0	0	57005	···
100001	2.166	11	7949	0	8	57005	···
100001	2.133	9	7955	0	1280008	57005	···
⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

Figure 1: S.M.A.R.T. dataset example (excerpt)

The neurons between layers are connected through weighted links, and the final output of the whole network designates the class of the instance.

3.5.4 *Voted Perceptron (VP)*

Voted perceptron, introduced by Freund and Schapire [18], is a classifier which finds the vector that can linearly separate data into classes, provided that the margin between them is sufficiently large. The performance of this method is claimed to be close to that of SVM but with a faster training time.

3.6 Instance-based Learning

Instance-based learning or lazy learning [2] is the technique that stores the instances learned during the training phase and postpones any processing or computation until runtime when new instances need to be classified.

3.6.1 *Nearest Neighbor Classifier (NNC)*

Nearest neighbor classifier compares a new instance with those stored in the reference set. The new instance and the closest one in the reference set are assumed to be generated from the same class, which is the class assigned to the new instance. Euclidean distance is a function generally used to compute the distance between instances.

3.6.2 *K-Star*

K-Star, introduced by Cleary and Trigg [11], is a learning algorithm similar to nearest neighbor classifier. However, the measure for the similarity between the new instance and the references is calculated using an entropy function.

3.6.3 *Locally Weighted Learning (LWL)*

Locally weighted learning [5] classifies a new instance based on the weighted distance of the nearest neighbors. Specifically, the underlying algorithm builds naïve Bayes models based on the k-nearest neighbors of the new instance and use these models to compute a class probability of that instance [16].

4. PERFORMANCE COMPARISON

We used the open-source data mining tool WEKA[1] [21] to evaluate the performance of machine learning algorithms against data collected from hard disk drive failures. This section provides details of our evaluation including a description of the dataset, the experimental setup, the metrics used in the comparison, and a discussion of the obtained results. The original and preprocessed dataset including program source code and result data are made publicly available and can be downloaded from our website.[2]

[1]http://www.cs.waikato.ac.nz/∼ml/weka/index.html
[2]http://aqua.cs.uni-kl.de/HDDDataAnalysis/

4.1 Dataset Description and Preprocessing

Self-monitoring, analysis and reporting technology (S.M.A.R.T.) is an industry-standard hard disk drive technology embedded in the firmware which is being used in most modern hard drives. During the drive's operation, its internal parameters, e.g., read/write error rate, servo, and temperature, are recorded at regular intervals and can be accessed by the operating system.

The dataset used in our experiment is taken from 369 hard drives, 178 of which are good drives and 191 of which have failed during operation. This dataset has originally been used by Murray *et al.* [36] and has been made publicly available by the authors [25]. The total number of recorded parameters in this dataset is 64, including the class value which indicates whether the drive eventually failed. Figure 1 shows an example S.M.A.R.T. dataset, which can be viewed as multiple streams of recorded parameters. An observation (instance) contains one monitored value for each parameter. The whole dataset contains 68,411 instances. Similar to Murray *et al.* [36], we select 26 indicative parameters for our experiment, which are hours before failure, GList1–3, PList, Servo1–3, Servo5, Servo7–8, Servo10, ReadError1–3, ReadError18–20, and FlyHeight5–12. The excluded parameters are drive serial numbers, hours of operation, and other parameters whose values are constant throughout the monitoring period.

Before the learning algorithms are applied, the instances in the training set that represent failing and non-failing drives have to be separated so that they can be used to train different prediction models. However, if only the instances that were collected at the time of failures are used to train the failing model, this model will not be able to make predictions as it can recognize only the instances when the drives fail, not before they fail. Therefore, the instances that are collected before the failures should also be used to train the failing model. To determine how long this period should be, let us make two assumptions. First, we assume that the drive exhibits different characteristics throughout its life time. The prediction model that is trained with the instances collected from any operation period will be able to recognize them at runtime. Second, we assume that a sufficient warning time should be seven days before a drive fails. This will give system administrators enough time to prepare for the failure. As a result, the instances that were collected within seven days before the failures are used to train the failing model, while those that are collected before seven days are used to train the non-failing model.

4.2 Experimental Setup

We applied the machine learning algorithms implemented in WEKA version 3.7.5 to the preprocessed dataset and evaluated the performance of each technique in terms of the metrics described in the following section. The evaluation is carried out on a physical computer equipped with a quad-core Intel Xeon E31220 processor running at 3.10 GHz with 16 GB of RAM and with Ubuntu Server 12.04.2 LTS as operating system. We used Java version 1.7.0 (JVM v. 21.0-b17; JRE v. 1.7.0-b147). The initial and maximum amount of heap space available for the JVM are both set to 4 GB in each of the experiments. No other tasks were executed on the machine during the experiment run.

The configurations of the algorithms in the experiment are set to the default values, as the main focus of this paper is to compare the performance between algorithms. The exhaustive parameter tuning of all algorithms is computationally expensive and almost infeasible.

4.3 Evaluation Metrics

This section gives a short description of the metrics used to evaluate the performance of the machine learning algorithms for proactive failure detection. Table 1 shows the contingency table of the basic metrics and Table 2 describes mathematical formulas for calculating the derived metrics. For a full and extensive description of these metrics, please refer to Salfner *et al.* [39].

True positive (TP) is an instance of a failing drive that is correctly predicted as failing.

False positive (FP) is an instance of a good drive that is incorrectly predicted as failing.

True negative (TN) is an instance of a good drive that is correctly predicted as non-failing.

False negative (FN) is an instance of a failing drive that is incorrectly predicted as non-failing.

True positive rate (TPR) or recall is the number of correctly predicted failing instances over the number of all failing instances.

False positive rate (FPR) is the number of negative instances that are classified as positive over the number of positive instances.

Precision is the number of correctly classified positive instance over the number of all instances predicted as positive.

Accuracy is the number of correct predictions over the total number of predictions made.

F-measure is a metric that combines precision and recall into one value.

Receiver operating characteristic (ROC) curve is the curve that indicates the performance of an algorithm by plotting the true positive rates against the false positive rates for different algorithm configurations [15], e.g., thresholds. ROC curves are also used to evaluate the quality of machine learning algorithms [7].

Area under ROC curve (AUC) is the area of the graph which is covered by the ROC curve of an algorithm.

Training time is the time required to train the algorithm, given a certain number of training instances.

Prediction time is the time a trained machine learning algorithm needs to make a certain number of predictions.

Table 1: Contingency table

	Good drive	Failing drive
Predicted as good	True negative	False negative
Predicted as failing	False positve	True positive

Table 2: Evaluation metrics

Metric	Formula
True positive rate, recall	$\frac{\text{TP}}{\text{TP+FN}}$
False positive rate	$\frac{\text{FP}}{\text{FP+TN}}$
Precision	$\frac{\text{TP}}{\text{TP+FP}}$
Accuracy	$\frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$
F-measure	$\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

4.4 Results

The experiment in this paper is divided into two parts. The first part is the comparison of the prediction quality of each algorithm based on the same dataset. The second part is the comparison of the time needed for the algorithms to build prediction models during the training phase and making prediction at runtime.

4.4.1 Prediction Quality

To measure the prediction quality (in terms of the metrics described in Section 4.3), each algorithm is trained with the instances in the dataset. The trained algorithms are used to make predictions. The results are the average values over 10-fold cross-validation.

Figure 2 illustrates the ROC curves of the 21 algorithms. In the very high quality region, the algorithms that have approximately the same level of prediction quality are nearest neighbor classifier, random forest, C4.5, REPTree, RIPPER, PART, and K-Star. Among the best algorithms, the one that has the highest true positive rate with low false positive rate is NNC. However, when the false positive rate increases, NNC is outperformed by random forest. On the other hand, the algorithms that perform poorly are SL, SGD, SMO, SVM, and ZeroR, showing a prediction quality close to a random predictor.

The algorithms that exhibit only one or two points in the curve are the ones that do not produce class probability as output. In other words, the threshold that separates failing and non-failing instances is fixed and cannot be varied to adjust the trade-off between true positive and false positive rate. These algorithms are NNC, K-Star, OneR, ZeroR, SGD, SMO, MNB, and SVM.

Table 3 summarizes the prediction quality of all machine learning algorithms. The decision threshold for all classifiers is set to the point with the maximum F-measure. This algorithm configuration is used to obtain the values of the evaluation metrics.

By using the F-measure as the primary metric, the best algorithm appears to be the nearest neighbour classifier with F-measure 0.976, followed by random forest and C4.5 at 0.957 and 0.946, respectively. Nevertheless, the algorithms in the upper half of the table have F-measure values higher than 0.6, while four algorithms in the lower half score lower than 0.1. For the highest true positive rate, recall, accuracy, and F-measure, NNC outperforms all other algorithms. On the other hand, for the highest precision, SVM performs best, followed by NNC. However, if a low false positive rate is the most important metric, for example, when the cost of a false alarm is very high, SMO, and SVM might be the best choices as they can achieve the lowest possible FPR, namely 0, although the other metrics are not very high.

4.4.2 *Training and Prediction Time*

The measurement of training and prediction time of the algorithms is done by measuring the time the algorithms take to train and make predictions for 68,411 instances of data. This large number of instances is used so as to emphasize the difference between slow and fast algorithms, as shown in Figure 3. The results are the average of the training and prediction time across 100 runs, except for the results of LWL, K-Star, and SVM. Their results are the average across 10 runs since these algorithms require a very long training or prediction time.

Table 4 provides further time statistics for each algorithm. As can be seen, the fastest algorithms in terms of training time are those in the category of instance-based learning and simple classifiers, i.e., LWL, K-Star, and NNC, which take less than 0.01 seconds to train the model using 68,411 instances. The highest training time is required by SVM, which takes approximately 33 minutes, followed by simple logistic regression and multilayer perceptron at approximately four and three minutes, respectively. Moreover, SMO—which is an optimization of SVM—performs significantly faster than SVM by a factor of 13.

On the other hand, the prediction time required by the instance-based learning algorithms are very high. NNC needs seven minutes to make 68,411 predictions, while K-Star requires roughly three hours and LWL takes almost fifteen hours to predict all of them. Furthermore, SVM that requires the longest training time also needs as much time as NNC to make predictions. The prediction time of other algorithms are negligible compared to the five slowest algorithms and are almost not visible in Figure 3.

4.5 Discussion

This section discusses why some prediction algorithms are slow while the others are fast, why they achieve different prediction quality and which one should be chosen in specific practical applications.

4.5.1 *Experimental Results*

During the training process, the instance-based algorithms are very fast as they require virtually no computation and the learned instances only need to be stored in the database. However, during the prediction phase, these algorithms perform very slowly. LWL, which is the slowest algorithm during the prediction, has to build naïve Bayes models from the k-nearest neighbors of a test instance to make one prediction. This model construction dramatically slows down the prediction process. For NNC and K-Star, the distance between the test instance and all learned instances have to be computed. While Euclidean distance is used in NNC, an entropy function is used in K-Star and, as a result, requires more computation power than NNC.

ZeroR is a rule-based algorithm which performs very fast for both training and prediction phases. The reason of the fast processing is obvious as it only counts the number of instances in each class during the training and assigns the test instance to the majority class during the prediction. OneR, DT, PART, and RIPPER are other rule-based algorithms with different training and prediction speeds. These algorithms analyze the training instances and create rules according to their specific methods which causes the training time to vary. When a test instance has to be classified, it has to be checked against the rules — typically conditional statements — which is a very fast process.

MNB, NBC, and BN are classifiers that build probabilistic models from the learned instances. MNB creates the model by counting the occurrences of the instance features while NBC builds normal distribution models out of them. Thus, NBC is slower than MNB as the parameters of the models have to be estimated. Besides, BN has to build both the network structure and the probabilistic models, which results in a slower training time than the other two algorithms in this category. However, for the prediction, NBC needs longer time than the other two since the class probabilities have to be computed from continuous distributions.

C4.5, REPTree, and RF are fast algorithms but still require some training time due to the tree construction phase, which searches for the best splitting nodes. Nonetheless, the prediction times of these algorithms are very fast, similar to rule-based algorithms, since the splitting nodes also use conditional statements to classify instances.

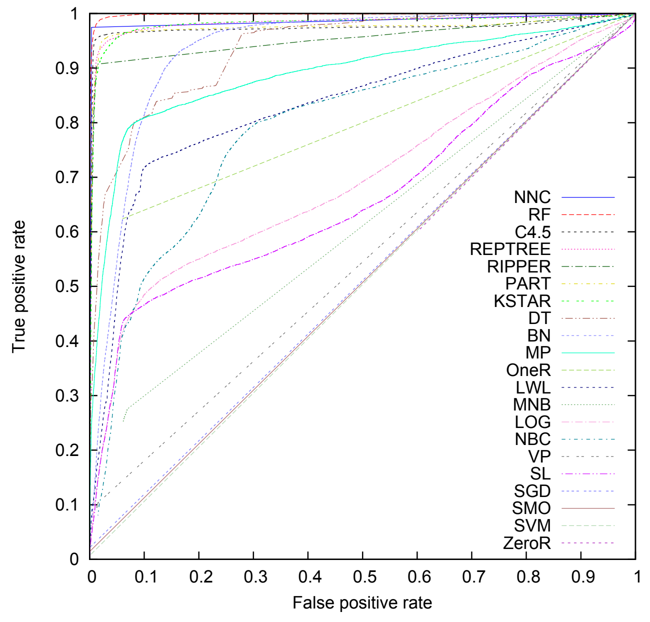
SVM, which uses a hyperplane to separate the instances into classes, has the slowest training time because it requires very high computational power for quadratic programming optimization. SMO and SGD use other approaches to solve the optimization problem which significantly improve SVM and reduce both training and prediction time.

For the prediction quality, NNC is the best algorithm as we assume that when the parameters are mapped to a multi-dimension space, they do not form distinct clusters but rather spread throughout the area. During the prediction phase, a test instance is then mapped to the point where it is closest to the one with the highest similarity. NNC can thus take advantage of this similarity and correctly classify that instance. In addition, RF, C4.5, and REPTree are decision trees that achieve very high prediction quality. However, as RF contains a number of decision trees and uses voting to decide the final outcome, it tends to perform better than C4.5 and REPTree.

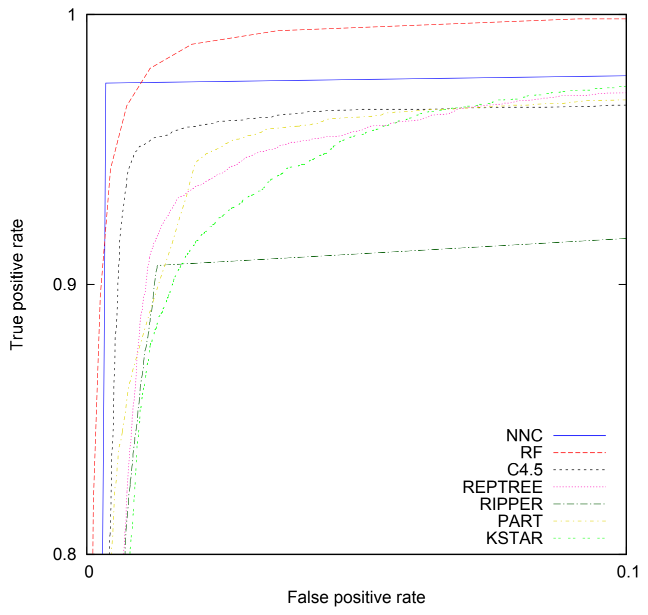
On the other hand, BN, MNB, and NBC do not achieve as good prediction quality as we expected. One of our assumptions is that as the data do not form clusters, these algorithms can not create probabilistic models that clearly separate the failing and non-failing instances. This assumption also applies to SVM, SMO, and SGD which use a hyperplane to classify instances and, therefore, perform more poorly than our expectation.

4.5.2 *Selection of Algorithms*

The decision of selecting an algorithm in practice depends on the application and the constraints. Two factors that should be considered are the prediction quality and time needed for training and prediction. If the algorithm is to be trained in an offline manner and later deployed under operation, the training time can be neglected, as it can be done separately in advance. If the algorithm will be used in online learning approaches, its training time needs to be very fast to be able to process new instances and to update the prediction model at runtime. However, the prediction time of the algorithm should be considerably fast to make a



(a) ROC curves



(b) ROC curves (excerpt from (a))

Figure 2: ROC curves

Table 3: Prediction quality of the selected algorithms ordered by F-measure

Algorithm	Abbr.	TPR	FPR	Precision	Recall	Accuracy	F-Measure	AUC
Nearest neighbor classifier	NNC	**0.974**	0.003	0.977	**0.974**	**0.993**	**0.976**	0.986
Random forest	RF	0.943	0.004	0.971	0.943	0.989	0.957	**0.998**
C4.5	C4.5	0.942	0.008	0.95	0.942	0.986	0.946	0.973
REPTree	REPTree	0.913	0.012	0.921	0.913	0.978	0.917	0.982
RIPPER	RIPPER	0.907	0.013	0.915	0.907	0.976	0.911	0.954
PART	PART	0.89	0.012	0.921	0.89	0.975	0.906	0.975
K-Star	K-Star	0.875	0.012	0.921	0.875	0.973	0.898	0.981
Decision table	DT	0.668	0.028	0.785	0.668	0.931	0.722	0.936
Bayesian network	BN	0.735	0.078	0.592	0.735	0.897	0.656	0.934
Multilayer perceptron	MP	0.585	0.032	0.739	0.585	0.917	0.653	0.89
OneR	OneR	0.624	0.06	0.616	0.624	0.897	0.62	0.782
Locally weighted learning	LWL	0.652	0.082	0.552	0.652	0.883	0.598	0.833
Multinomial naïve Bayes classifier	MNB	0.252	0.061	0.388	0.252	0.846	0.305	0.603
Logistic regression	LOG	0.124	0.012	0.618	0.124	0.872	0.206	0.697
Naïve Bayes classifier	NBC	0.118	0.022	0.457	0.118	0.863	0.188	0.789
Voted perceptron	VP	0.094	0.013	0.527	0.094	0.867	0.16	0.544
Simple logistic regression	SL	0.08	0.008	0.598	0.08	0.87	0.14	0.665
Stochastic gradient descent	SGD	0.022	0.001	0.792	0.022	0.868	0.044	0.511
Sequential minimal optimization	SMO	0.015	**0**	0.86	0.015	0.868	0.029	0.507
Support vector machine	SVM	0.007	**0**	**0.984**	0.007	0.867	0.014	0.503
ZeroR	ZeroR	0	**0**	0	0	0.866	0	0.5

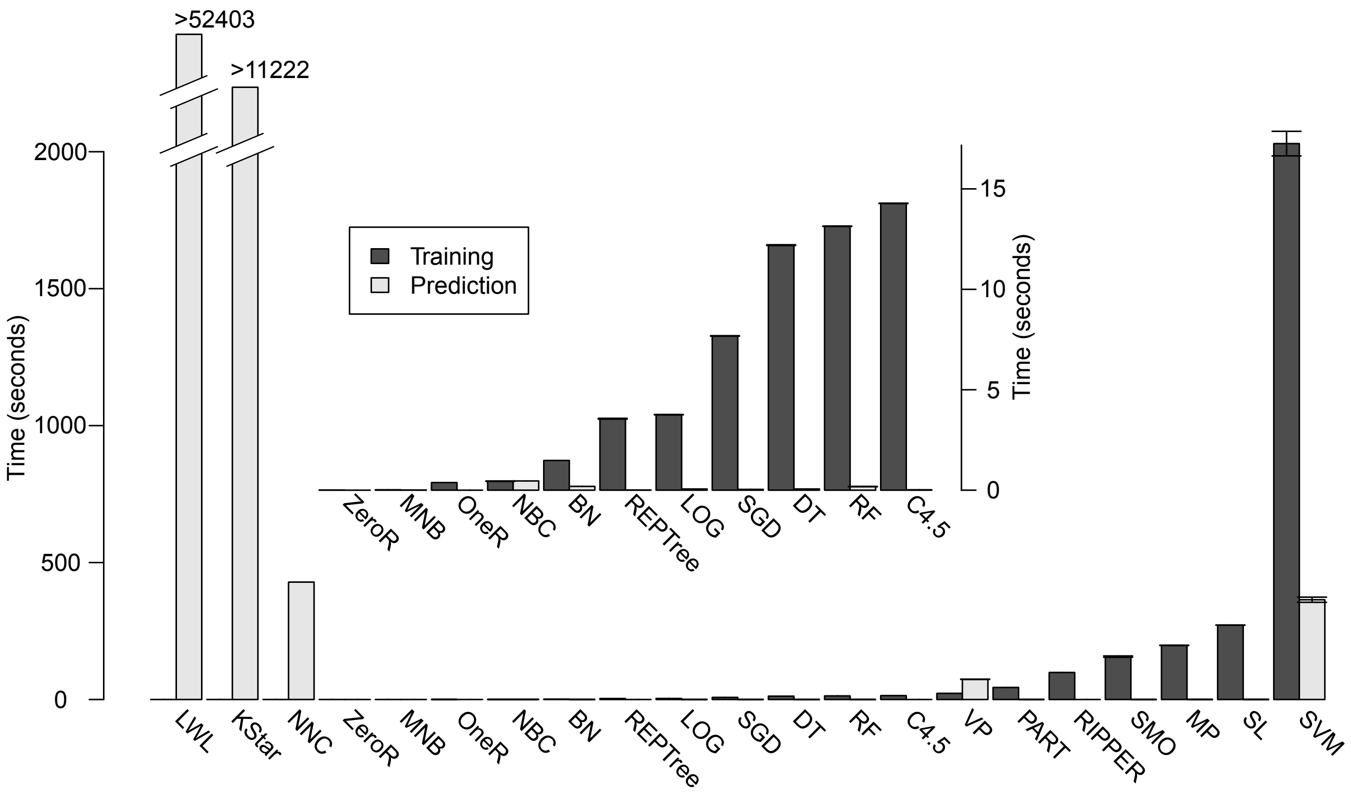


Figure 3: Mean training and prediction times. The inner barplot is an enlarged version of the times for selected algorithms.

Table 4: Training and prediction time statistics averaged across 100 runs. The algorithms with 10 runs are denoted by *.

Algorithms	Training (seconds)					Prediction (seconds)				
	Mean	(95% CI)	Q_1	Q_2	Q_3	Mean	(95% CI)	Q_1	Q_2	Q_3
LWL*	0.01	(±<0.01)	0.01	0.01	0.01	52403.33	(±3907.99)	48588.27	49598.24	54346.42
K-Star*	0.01	(±<0.01)	0.01	0.01	0.01	11446.98	(±177.08)	11222.57	11367.73	11707.61
NNC	0.01	(±<0.01)	0.01	0.01	0.01	428.64	(±0.16)	428.01	428.42	429.10
ZeroR	0.01	(±<0.01)	0.01	0.01	0.01	<0.01	(±<0.01)	<0.01	<0.01	<0.01
MNB	0.02	(±<0.01)	0.01	0.01	0.01	0.01	(±<0.01)	0.01	0.01	0.01
OneR	0.38	(±<0.01)	0.37	0.37	0.38	<0.01	(±<0.01)	<0.01	<0.01	<0.01
NBC	0.45	(±0.01)	0.45	0.45	0.46	0.46	(±<0.01)	0.46	0.46	0.46
BN	1.48	(±<0.01)	1.46	1.47	1.49	0.19	(±<0.01)	0.18	0.19	0.19
REPTree	3.56	(±0.02)	3.53	3.56	3.60	0.01	(±<0.01)	0.01	0.01	0.01
LOG	3.76	(±0.01)	3.71	3.73	3.80	0.06	(±<0.01)	0.06	0.06	0.06
SGD	7.68	(±0.01)	7.66	7.68	7.70	0.04	(±<0.01)	0.03	0.04	0.04
DT	12.20	(±0.02)	12.11	12.21	12.28	0.06	(±<0.01)	0.06	0.06	0.06
RF	13.14	(±0.01)	13.12	13.14	13.16	0.18	(±0.01)	0.16	0.20	0.20
C4.5	14.28	(±0.01)	14.24	14.27	14.32	0.02	(±<0.01)	0.02	0.02	0.02
VP	22.48	(±0.06)	22.33	22.46	22.66	73.80	(±0.51)	72.46	73.98	75.30
PART	43.97	(±0.16)	43.73	43.81	43.90	0.09	(±<0.01)	0.09	0.09	0.09
RIPPER	98.72	(±0.26)	98.10	98.66	99.28	0.03	(±<0.01)	0.03	0.03	0.03
SMO	156.60	(±2.48)	147.24	148.85	171.77	0.04	(±<0.01)	0.04	0.04	0.04
MP	197.76	(±0.31)	196.67	197.20	198.54	0.27	(±<0.01)	0.27	0.27	0.28
SL	271.65	(±0.33)	270.82	272.13	272.72	0.30	(±<0.01)	0.29	0.30	0.30
SVM*	2029.64	(±44.85)	2001.01	2021.05	2072.45	364.26	(±9.46)	354.86	365.15	375.18

prediction based on the most recent instance, before a new one arrives. When the training and prediction times are not the primary factor, the prediction quality becomes the first factor to be considered. The true positive rate should be taken into account when the cost of missing a failure is high. When the cost of a false alarm is high, the false positive rate should be considered. The overall prediction quality can be determined by F-measure value, which combines both true positive rate and false negative rate.

From our experiment, the algorithms which are suitable for applications that require high prediction quality without time constraint are nearest neighbor classifier, random forest, C4.5, REPTree, RIPPER, PART, and K-Star. The algorithms applicable for online learning approaches are Bayesian network and OneR since they require both short training and prediction times while maintaining relatively high prediction quality. When a low or closest-to-zero false alarm rate is desired, SMO, and SVM appear to be the best options, even though their true positive rates are quite low in comparison to other algorithms.

Nonetheless, the algorithms with comparable prediction qualities may perform differently in practical applications. Even though NNC has the highest true positive rate in the lower range of false positive rate, other algorithms may outperform it when they are deployed. Therefore, the algorithms should be specifically evaluated against a certain task before being selected for real use.

4.6 Threats to Validity

Threats to internal validity. One assumption we made to separate failing instances from non-failing ones is the seven-day time frame before failure. We assume that the signs of a failure are observable during this period independently from the failure type. However, this period could be shorter, or closer to the failure than seven days, and the characteristic of the captured data is actually from the non-failing drives. Consequently, the failure model could be tampered with good instances, which could cause the results to have high false positive rates.

Moreover, the algorithms in our experiment are set to their default configurations provided by WEKA. Some algorithms that allow parameter tuning or internal modification, such as SVM with kernel trick, may perform better when their parameters are properly set to match the property of the data. Thus, our results are valid only for specific settings of the tested algorithms.

Threats to external validity. The experiment carried out in this paper is based on a single dataset. Some characteristics embedded in the data may give advantages to some algorithms over the others. This may cause biases in the results with overly optimistic or pessimistic prediction qualities of some algorithms.

5. CONCLUSIONS AND FUTURE WORK

Proactive failure detection is an approach which aims to foresee a pending failure and to issue a warning or initiate a recovery action before the failure will cause damage to the system. Machine learning techniques are a suitable approach to proactively detect failures by analyzing available information from system monitoring. The signs or system parameters that exhibit a healthy state of the system are used to build a non-failing model while the signs occurring before failures are used to build a failing one. At runtime, the signs observed from the system are compared against these two models and a corresponding prediction is made whether the system is deemed to fail in the near future.

In this paper, we evaluated 21 machine learning algorithms against a publicly available failure dataset collected from hard disk drives. Two aspects of the algorithms, which are prediction quality as well as training and prediction times, are compared. The results showed that each algorithm has different advantages depending on the application constrains. Choosing the best algorithm for a specific application needs to take into account both the prediction quality and the training and prediction times.

There are still open questions to be explored in proactive failure detection, such as lead time which indicates how much time is left before a failure will occur. In our future work, we aim to improve lead time prediction by providing accurate values indicating when the failure will occur so that the recovery procedure can be carefully planned and efficiently executed.

6. REFERENCES

[1] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. A comparison of machine learning techniques for phishing detection. In *Proc. Anti-Phishing Working Group's 2nd Annual eCrime Researchers Summit*, eCrime '07, pages 60–69. ACM, 2007.

[2] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] J. Alonso, J. Torres, J. L. Berral, and R. Gavalda. Adaptive on-line software aging prediction based on machine learning. In *Proc. 40th IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN '10)*, pages 507–516, July 2010.

[4] A. Andrzejak and L. Silva. Using machine learning for non-intrusive modeling and prediction of software aging. In *Proc. IEEE Network Operations and Management Symposium (NOMS '08)*, pages 25–32. IEEE, Apr. 2008.

[5] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[6] L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proc. 19th Int'l Conf. on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, August 2010. Springer.

[7] A. P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[8] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[9] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proc. 23rd Int'l Conf. on Machine Learning (ICML '06)*, pages 161–168. ACM, 2006.

[10] K. Chan, T.-W. Lee, P. A. Sample, M. H. Goldbaum, R. N. Weinreb, and T. J. Sejnowski. Comparison of machine learning and traditional classifiers in glaucoma diagnosis. *IEEE Transactions on Biomedical Engineering*, 49(9):963–974, 2002.

[11] J. G. Cleary and L. E. Trigg. K*: An instance-based

learner using an entropic distance measure. In *Proc. 12th Int'l Conf. on Machine Learning (ICML '95)*, pages 108–114. Morgan Kaufmann, 1995.

[12] W. W. Cohen. Fast effective rule induction. In *Proc. 12th Int'l Conf. on Machine Learning (ICML '95)*, pages 115–123. Morgan Kaufmann, 1995.

[13] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[14] S. Dreiseitl, L. Ohno-Machado, H. Kittler, S. Vinterbo, H. Billhardt, and M. Binder. A comparison of machine learning methods for the diagnosis of pigmented skin lesions. *Journal of Biomedical Informatics*, 34(1):28–36, 2001.

[15] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[16] E. Frank, M. Hall, and B. Pfahringer. Locally weighted naive bayes. In *Proc. 19th Conf. on Uncertainty in Artificial Intelligence (UAI '03)*, pages 249–256, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[17] E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proc. 15th Int'l Conf. on Machine Learning (ICML 98)*, pages 144–151. Morgan Kaufmann, 1998.

[18] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Proc. 11th Annual Conf. on Computational Learning Theory (COLT '98)*, pages 209–217. ACM, 1998.

[19] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.

[20] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[22] G. Hamerly and C. Elkan. Bayesian approaches to failure prediction for disk drives. In *Proc. 18th Int'l Conf. on Machine Learning (ICML '01)*, pages 202–209. Morgan Kaufmann, 2001.

[23] S. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall Int'l Editions Series. Prentice Hall, 1999.

[24] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–90, 1993.

[25] G. F. Hughes. *S.M.A.R.T. Dataset*, Feb. 2013. http://cmrr.ucsd.edu/people/hughes/smart/dataset/.

[26] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 51:2002, 2002.

[27] A. M. Kibriya, E. Frank, B. Pfahringer, and G. Holmes. Multinomial naive bayes for text categorization revisited. In G. Webb and X. Yu, editors, *AI 2004: Advances in Artificial Intelligence*, volume 3339 of *LNCS*, pages 488–499. Springer Berlin Heidelberg, 2005.

[28] R. Kohavi. The power of decision tables. In *Proc. 8th European Conf. on Machine Learning (ECML '95)*, volume 912 of *LNCS*, pages 174–189. Springer, 1995.

[29] N. Landwehr, M. Hall, and E. Frank. Logistic model trees. 95(1-2):161–205, 2005.

[30] S. le Cessie and J. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

[31] Y. Liang, Y. Zhang, H. Xiong, and R. K. Sahoo. Failure prediction in IBM BlueGene/L event logs. In *Proc. 7th IEEE Int'l Conf. on Data Mining (ICDM '07)*, pages 583–588. IEEE, 2007.

[32] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40:203–228, 2000.

[33] D. Lowd and P. Domingos. Naive bayes models for probability estimation. In *Proc. 22nd Int'l Conf. on Machine Learning (ICML '05)*, pages 529–536. ACM, 2005.

[34] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster. An investigation of machine learning based prediction systems. *Journal of Systems and Software*, 53(1):23–29, 2000.

[35] A. Metzger, O. Sammodi, and K. Pohl. Accurate proactive adaptation of service-oriented systems. In *Assurances for Self-Adaptive Systems*, volume 7740 of *LNCS*, pages 240–265. Springer, 2013.

[36] J. F. Murray, G. F. Hughes, and D. Schuurmans. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning research*, 6:816, 2005.

[37] J. C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[38] J. R. Quinlan. *C4.5: Programs for machine learning*, volume 1. Morgan Kaufmann, 1993.

[39] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):1–42, 2010.

[40] R. Sasisekharan, V. Seshadri, and S. Weiss. Proactive network maintenance using machine learning. In *Proc. IEEE Global Telecommunications Conf. (GLOBECOM '93)*, pages 217–222, 1993.

[41] M. Sumner, E. Frank, and M. Hall. Speeding up logistic model tree induction. In *9th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD '05)*, pages 675–683. Springer, 2005.

[42] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18:77–95, 2002.

[43] S. M. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets and machine learning classification methods. *Readings in machine learning*, pages 177–183, 1990.

[44] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, Oct. 2006.