SOFTWARE DEFECT PREDICTION USING MACHINE LEARNING TECHNIQUES

Humaira Khalid[1], Mansoor Hai[2*], M. Sharif[3], Aijaz Panhwar[2], Zainulibad[2], Mehtab Ahmed[2], Aijaz Solangi[2], Zain Solangi[2], Baitullah[2]

[1] PCSIR Head Office Islamabad, Pakistan

[2] PCSIR Laboratories Complex, Karachi, Sindh, Pakistan

[3] Benazir Bhutto Shaheed University Lyari, Karachi, Sindh, Pakistan

*Corresponding Author: mansoorhai@yahoo.com

ABSTRACT: Software flaws can result in large financial losses, decreased user' happiness, and a general decrease in the dependability of software systems. Therefore, early in the software development lifecycle defect, detection and mitigation are critical. This paper provides an overview of machine learning-based software defect prediction. Exploring the use of machine learning algorithms to forecast and categorize software problems based on historical data and other software metrics is the goal. In addition to discussing the difficulties in defect prediction, this study provides a thorough analysis of the most recent machine learning methods used to this problem. Additionally, it sheds light on the feature selection, evaluation criteria, and data preprocessing methods frequently used in software defect prediction studies. The article concludes with a comparison of various machines learning methods and how well they perform in forecasting software flaws, highlighting the advantages and disadvantages of each method. The results of this study help to advance our understanding of software defect prediction using machine learning and give researchers and practitioner's advice on how to select the best tools for their individual needs.

Key words: Algorithms Defect, Evaluation, Machine Learning, Techniques, Software.

(Received 02.10.2025 Accepted 15.12.2025)

INTRODUCTION

Software flaws can have detrimental effects, such as lost revenue, lowered user happiness, and weakened system dependability. In the software development process, early defect detection and mitigation are crucial. Software Defect Prediction is part of the software development life cycle in which we predict the fault using a Machine Learning approach with historical data [1].An overview of software fault prediction using machine learning approaches is the goal of this study. Exploring how machine learning algorithms can be used to forecast and categorize software problems based on historical data and different software indicators is the goal. The study examines the most recent machine learning methods applied to defect prediction as well as the difficulties connected with defect prediction. It also covers the methods for data preprocessing, feature selection, and assessment metrics frequently used in software defect prediction research. The paper concludes with a comparative analysis of different machine learning algorithms and their performance in predicting software defects, highlighting their strengths and limitations. The findings of this study contribute to advancing the understanding of software defect prediction using machine learning and offer valuable guidance for researchers and practitioners in selecting suitable techniques for their specific needs.

Background: Software defects are inevitable in software development and can cause a variety of problems, including system failures, security flaws, and user dissatisfaction. From an external perspective, a defect is the violation or failure of the framework/system to accomplish specific capacities [2, 3]. Traditional defect detection approaches heavily rely on manual testing, which is time-consuming, expensive, and may not provide comprehensive coverage. Detecting and resolving defects early in the software development lifecycle is crucial to ensure the delivery of high-quality software products. Promising approaches for predicting software defects now include machine learning techniques. Machine learning algorithms can uncover patterns and associations that may be used to detect possible flaws in software systems by utilizing historical data and different software metrics. By automating the defect prediction process, this method has the advantage of giving developers insightful data they can use to set priorities and assign resources for defect resolving. In recent years, there has been a lot of interest in the application of machine learning techniques for software fault prediction. To boost the precision and efficacy of defect prediction models, researchers and practitioners have investigated a variety of algorithms, data preprocessing techniques, and evaluation measures. This study attempts to give a thorough overview of the state-of-the-art in software defect prediction using machine learning techniques, emphasizing the strides made in this area and the issues that still need to be resolved. Researchers and practitioners can obtain insight into the possible advantages and constraints of these strategies by comprehending the history and present state of software defect prediction using machine learning. This information can assist in the construction of more precise and effective defect prediction models as well as the formulation of successful defect management plans for software development projects.

Motivation: This paper's motivation comes from the growing significance of software defect prediction in the field of software engineering. Delivering high-quality software products requires early problem detection and repair in the software development process. The conventional manual testing methods can be costly, time-consuming, and may not offer complete coverage. The most critical stage of any software, which necessitates comprehensive testing, is software defect identification, which is the most significant aspect of the SDLC [4].The automation of the fault prediction process and the provision of insightful data to software engineers are both possible outcomes of machine learning techniques. Machine learning algorithms can uncover trends and associations that could potentially point to software system flaws by examining historical data and software metrics. Nevertheless, there is a need for a thorough assessment of the most cutting-edge approaches, difficulties, and best practices in this field despite the growing interest in applying machine learning for defect prediction. This paper's main goal is to close the information gap by offering a thorough examination of machine learning-based software fault prediction. This paper seeks to assemble the body of knowledge and developments in the area by researching and evaluating the available literature. Additionally, it seeks to offer academics and professionals insightful information on how to choose the best machine learning algorithms, data pretreatment methods, feature selection strategies, and assessment metrics for efficient defect prediction. Additionally, by comparing several machine learning methods, this research intends to show their advantages, disadvantages, and efficacy in identifying software problems. This analysis can help academics and professionals select the best algorithm for their unique needs.

The overall goal of this study is to increase knowledge of software defect prediction using machine learning and to serve as a useful tool for researchers, practitioners, and decision-makers involved in software development and quality assurance.

Objectives: This research paper aims to:

1. Give an overview of the prediction of software defects and their significance in software engineering.
2. Research and assess the effectiveness of machine learning techniques for forecasting software defects.
3. Examine the challenges and limitations of machine learning for defect prediction and consider alternate approaches.
4. Discuss the effects of data preparation techniques on defect prediction models.
5. For software defect forecasting, look at feature selection methodologies.
6. Methods of machine learning for defect prediction are compared and contrasted.
7. Give advice on the standards to use when evaluating defect prediction models.
8. The benefits and drawbacks of various machine learning techniques should be highlighted.
9. Determine potential areas for future research in software defect prediction using machine learning.

The paper's overall goal is to increase software fault prediction using machine learning by offering comprehension, insights, and new information.

Software Defect Prediction

Definition and Importance: Software defect prediction is the process of identifying and categorizing probable flaws in software systems using predictive models. In order to effectively manage resources and concentrate on the key regions for defect identification and resolution, it seeks to proactively identify areas of code that are more likely to have flaws.

Predicting software failures is crucial because it can stop problems from spreading to later phases of the software development lifecycle. Early detection of potential flaws allows developers to fix them, minimizing their effects on system dependability, user happiness, and total project costs. Defect prediction also helps to increase maintenance efforts, enhance software quality, and streamline software testing procedures.

Challenges and Limitations: There are difficulties and restrictions with software fault prediction. Some of the major difficulties include:

a. Unbalanced datasets: Software defect datasets frequently show class imbalance, where the proportion of defective instances to non-defective instances is much lower. Machine learning algorithms' performance may be impacted by this mismatch, producing inaccurate results.
b. Feature selection: It's essential to choose pertinent software metrics or features for fault prediction. However, it might be difficult to pick out the most useful aspects from the huge array of data that are available. To get around this problem, feature selection techniques are used.
c. Data quality: The accuracy of the predictive models might be impacted by the quality of the historical data utilized for defect prediction. Uncertainty can be

introduced and have an impact on the performance of the models due to incomplete or inconsistent data, noise, and missing values.

d. Over-fitting: Over-fitting is a problem with machine learning models where the model gets too specialized to the training data and is unable to generalize successfully to new, untainted data. When used with real-world settings, over-fitting might result in subpar predicting accuracy.

Approaches to Defect Prediction: For predicting software defects, a number of strategies have been used, such as:

Supervised learning: This approach involves which involves building machine learning models using past data that has been labeled as either defective or not. The models use the labeled data's patterns to predict new, unlabeled instances.

Un-supervised learning: When there is a lack of labeled data or it is inaccessible, unsupervised learning approaches are applied. These methods seek to find patterns and abnormalities in data without being aware of fault labels beforehand.

Semi-supervised learning: Predictive models are created using this method, which mixes labeled and unlabeled data. Labeled instances reveal the presence of defects, whereas unlabeled instances aid in the identification of underlying patterns.

Transfer learning: To enhance fault prediction in the target domain, transfer learning uses knowledge from a source domain (such as a different project). It is useful when the target domain has few labeled data points. The choice of an appropriate strategy is influenced by the accessibility of labeled data, the nature of the software project, and the precise defect prediction objectives. In summary, software defect prediction is extremely important for raising software quality and minimizing the effects of errors. Addressing issues such uneven datasets, feature selection, poor data quality, and over-fitting are necessary. Effective defect prediction can be achieved using a variety of techniques, such as supervised learning, unsupervised learning, semi-supervised learning, and transfer learning.

Machine learning techniques for defect prediction: Due to their capacity to recognize patterns and relationships in past data, machine learning techniques have been extensively used in software fault prediction. Here, essential elements of machine learning for predicting defects, such as data preprocessing, feature selection, and the use of several classification techniques are reviewed.

Data Preprocessing: The quality and usability of the data for machine learning algorithms are ensured through data preprocessing, a critical step in defect prediction. There are several common preprocessing methods:

Data cleaning: Getting rid of duplicate instances, dealing with missing values, and dealing with outliers to maintain data integrity.

Data normalization: To avoid features with bigger values dominating the model, scale the data to a common range (for example, 0 to 1).

Data balancing: Addressing the issue of class inequality with methods like SMOTE (Synthetic Minority Over-sampling Technique), which involve either oversampling the minority class or under-sampling the dominant class.

Dimensionality reduction: Fewer characteristics are used in order to reduce computing complexity and noise risk. Algorithms for feature selection or Principal Component Analysis (PCA) can be used.

Feature Selection: Finding the most pertinent and instructive features for defect prediction are the goal of feature selection. This process aids in reducing dimensionality, enhancing model performance, and improving interpretability. Typical feature selection methods include:

Statistical methods: Using statistical tests to evaluate the usefulness of features, such as chi-square analysis, correlation, and information gain.

Wrapper methods: Making use of search techniques like backward elimination, forward selection, or genetic algorithms to assess the prediction strength of subsets of features.

Embedded methods: utilizing feature selection strategies built into machine learning algorithms, like regularization techniques (like L1 or L2 regularization).

Classification Algorithms: Defect prediction can be accomplished using a variety of machine learning algorithms. Numerous examinations demonstrate that the technique is a significant strategy to study the classification of software defects [5]. The exact properties of the data and the application's needs determine which algorithm is used. A few often employed algorithms are as follows:

Decision Trees: Tree-based models that divide the data into categories based on features to produce a decision tree structure.

Random Forests: This ensemble technique combines different decision trees to increase prediction accuracy.

Support Vector Machines (SVM): SVM is a supervised ML model mostly applied to data with two classes as output [6]. An algorithm for binary classification that locates a hyper plane to divide data points. Apart from linear classification, they can also perform non-linear classification efficiently. [7]

Naive Bayes: It is a probabilistic classifier based on Bayes theorem, which works on the primary assumption that features are conditionally independent [8].

Neural Networks: MLP can be used to address nearly any problem, including pattern recognition, Interpolation [9].Neurons are interconnected nodes in multilayered networks called neural networks that are used to learn complex patterns.

Ensemble Methods: Combining different models, such AdaBoost or Gradient Boosting, to produce a more accurate predictive model.

Several metrics, including accuracy, precision, recall, F1-score, ROC curve, and AUC, can be used to assess how well these algorithms work. Selecting the best algorithm for a certain defect prediction task requires testing out a variety of algorithms and comparing their effectiveness. The performance and generalizability of the selected algorithm can also be enhanced by using techniques for hyper-parameter adjustment and model optimization. In conclusion, feature selection, data preprocessing, and the application of multiple classification algorithms are all part of machine learning techniques for defect prediction. Data preparation ensures data quality. The data's structure and the task's unique criteria for defect prediction determine the selection of the best techniques.

Evaluation Metrics: The effectiveness of defect prediction models is evaluated in large part through evaluation metrics. They aid in evaluating how well, accurately, and consistently the models anticipate software faults. Following are some typical evaluation measures for fault prediction:

Accuracy: A key indicator of how well forecasts are made generally is accuracy. It is figured out as the proportion of accurately predicted instances (including true positives and true negatives) to all instances. When working with datasets that are unbalanced and where the majority class predominates, precision may not be enough on its own.

Precision: When comparing all anticipated positive instances, precision is defined as the percentage of accurately predicted positive occurrences (also known as faulty instances). It focuses on cutting down on false positives, which happen when occurrences that aren't actually defective are mistakenly labelled as such. A low rate of false positives is suggested by a high precision.

Recall (Sensitivity or True Positive Rate): Measured by recall, the percentage of accurately predicted positive cases (defective instances) among all actual positive instances. False, negatives incidents when defective instances are mistakenly labeled as non-defective are minimized as a main goal. A low percentage of false negatives is suggested by a good recall.

F1-Score: A balanced evaluation statistic that takes into accounts both false positives and false negatives are provided by the F1-score, which is the harmonic mean of precision and recall. It is determined by multiplying 2 by (precision * recall) / (precision + recall). When there is an unbalance between recall and precision, the F1-score is helpful.

Receiver Operating Characteristic (ROC) Curve and area Under the Curve (AUC): The true positive rate (recall) versus the false positive rate trade-off is graphically represented by the ROC curve. It displays how well a classifier performs at different categorization thresholds. The AUC denotes the region beneath the ROC curve and offers a single value that summarizes the overall effectiveness of the classifier. A better-performing model has a greater AUC. These are but a handful of instances of evaluation metrics that are applied to defect prediction. Other metrics, like specificity, Matthew's Correlation Coefficient (MCC), or balanced accuracy, might also be used, depending on the task's specific requirements. When choosing proper assessment metrics, it is crucial to take into account the features of the dataset and the precise objectives of the defect prediction task. The selection should be in line with the priorities and specifications of the software development process. Different metrics offer different perspectives on model performance.

Experimental setup and results: The experimental setup for predicting software defects entails the choosing of datasets, the machine learning algorithms, the setting of parameters, and the assessment of model efficacy. Here is a description of the experimental design and the way the findings were presented:

Dataset Selection: A suitable dataset for the experiment is chosen, consisting of historical software data with instances labeled to indicate whether they are flawed or not. The dataset needs to be adequate in scope and indicative of the target software system's software metrics.

Data Split: Training and testing sets are created from the dataset. The testing set is used to assess the performance and generalization skills of the machine learning models, whereas the training set is used to train the models. To ensure that the distribution of defective and non-defective cases is maintained in both sets, the data can be divided randomly or using certain methods such stratified sampling.

Machine Learning Algorithms: Several machine learning algorithms are chosen for the experiment based on how well they perform fault prediction tasks. Decision trees, random forests, support vector machines, neural networks, or other previously discussed methods may be among them. The methods are configured with the proper parameters, such as the regularization parameter in support vector machines or the number of trees in a random forest, when they are implemented.

Performance Evaluation: The testing set is used to assess the trained models. To evaluate the effectiveness of the models, many evaluation metrics are produced, including accuracy, precision, recall, F1-score, ROC curve, and AUC. Depending on the needs, additional metrics like specificity or MCC may also be computed.

Comparative Analysis: The outcomes of several machine learning algorithms are contrasted and examined. To determine which algorithms produce the best precision/recall trade-off or the highest accuracy, performance measurements are evaluated. Taking into account elements like interpretability, computational complexity, and scalability, the benefits and drawbacks of each algorithm are explored.

Statistical Analysis: To ascertain the data' significance, statistical analysis may be used. The validity of the observed differences across models can be evaluated using methods like hypothesis testing, confidence intervals, or statistical significance tests.

RESULTS AND DISCUSSION

The important discoveries and ideas are highlighted in a detailed discussion of the experimental data. Analysis is done on the variables affecting performance, such as dataset characteristics, method choice, and parameter tweaking. Discussions of any intriguing findings or unexpected results offer explanations or suggest possible directions for additional research.

Limitations: Any restrictions on the experimental design or the findings were addressed and acknowledged. These restrictions could be related to the dataset's representativeness, biases, presumptions made during the experiment, or limits imposed by the selected machine learning techniques. An in-depth knowledge of the experiments that were carried out, how well the machine learning models performed, and the consequences of the findings are intended to be provided through the presentation of the experimental setup and outcomes. It helps researchers and professionals to evaluate the efficacy of various strategies and make defensible choices regarding the incorporation of defect prediction techniques in software engineering processes.

Comparative Analysis: Comparative analysis in software defect prediction compares the effectiveness of several machine learning algorithms or methodologies in order to determine the best strategy for doing so. A comprehensive analysis of previous research on software failure, emphasizing metrics, procedures and public-private data sets, was published by C Cata *et al*. [10].

The steps involved in the comparative analysis are outlined below:

Selection of Algorithms: For comparison, a collection of machine learning techniques is chosen, including decision trees, random forests, support vector machines, naive Bayes, neural networks, and ensemble approaches. These algorithms are selected for defect prediction tasks based on their applicability, their popularity in the literature, or their potential to achieve high accuracy or balanced precision-recall trade-offs.

Experimental Setup: For a fair comparison, the experimental setup for each method uses the same dataset and assessment measures. The dataset is divided into training and testing sets, with the same partitions used for both algorithm training and evaluation. To improve performance, the hyper-parameters of the algorithms can be adjusted using strategies like grid search or cross-validation.

Performance Evaluation: The effectiveness of each algorithm is assessed using the chosen evaluation measures, such as accuracy, precision, recall, F1-score, ROC curve, and AUC. Each algorithm's outcomes are calculated and noted in order to derive a numerical assessment of their predictive power.

Analysis of Results: Analyses and comparisons are made of the outcomes of various algorithms. To ascertain which algorithms attain the best accuracy or offer the best precision/recall trade-off, key performance measures are analyzed. Using the proper statistical tests, it is possible to determine whether any detected changes are statistically significant.

Strengths and Limitations: Strengths and Drawbacks Based on the outcomes, the advantages and disadvantages of each method are highlighted. The advantages could be excellent accuracy, noise resistance, or the capacity to manage unbalanced datasets. It's possible that the restrictions relate to problems with interpretability, computational complexity, or sensitivity to parameter settings.

Interpretability and Explain ability: The algorithms' readability and explicability are taken into account. Some algorithms, like as naïve Bayes or decision trees, give clear and understandable models that might shed light on the variables influencing defect prediction. Other algorithms may offer great predicted accuracy but lack interpretability, such as neural networks or ensemble approaches. Discussion is had regarding the compromise between performance and interpretability.

Generalizability: The effectiveness of the algorithms is evaluated by comparing them across various datasets or software projects. Algorithms with a higher degree of generalizability regularly outperform on a variety of datasets.

Practical Considerations: The availability of libraries or frameworks, scalability, computational needs, and ease of implementation are only a few examples of practical factors that are taken into account. These elements have an impact on the applicability and viability of the algorithms in actual software development settings.

Recommendations: The best suited algorithm(s) for software defect prediction are suggested based on the comparison study. The formulation of these recommendations takes into accounts the advantages and disadvantages of each algorithm as well as their trade-offs. It is also important to consider the software development project's particular context, requirements, and restrictions. The comparison analysis assists academics and industry professionals in selecting the best machine learning algorithms for defect prediction. It gives information about the effectiveness, advantages, disadvantages, interpretability, and generalizability of various methods, assisting in the selection of efficient defect prediction methods for use in software engineering processes.

Future Directions and Research Challenges: The future directions and research problems in the subject of software defect prediction can influence its development. Here are some probable concentration points:

Incorporating Advanced Machine Learning Techniques: Utilizing Cutting-Edge Machine Learning Methodologies: Future studies may examine the use of cutting-edge machine learning methods to anticipate software defects, including deep learning, reinforcement learning, and transfer learning. These methodologies may find novel patterns and relationships in software data and have demonstrated promising outcomes in a number of disciplines.

Handling Imbalanced Data: Imbalanced datasets pose a challenge in defect prediction, as the majority class may overshadow the minority class, leading to biased models. Future research can focus on developing effective techniques for handling imbalanced data, such as advanced sampling methods, cost-sensitive learning approaches, or ensemble techniques specifically designed for imbalanced datasets.

Feature Engineering and Selection: In order to foresee defects, feature engineering is essential. Future studies could look into novel approaches to extracting and visualizing software metrics or introducing domain-specific expertise into the feature engineering workflows. Research may also concentrate on creating automated or intelligent feature selection techniques that can pinpoint the most pertinent and instructive features for defect prediction.

Multi-Objective Optimization: In order to maximize accuracy while reducing false positives or false negatives, defect prediction frequently involves numerous competing goals. To identify a set of solutions that represent various trade-offs between these objectives, future research can investigate multi-objective optimization approaches. Decision-makers may then have a variety of options based on their own goals and needs.

Explainable AI in Defect Prediction: In order for stakeholders to understand and have confidence in the prediction models, interpretability and explainability are essential. The development of explainable AI methods that offer open and clear insights into the elements influencing defect prediction can be the focus of future study. This may enhance the use and acceptability of defect prediction models in actual applications.

Context Aware Defect Prediction: Consideration of contextual data, such as the development environment, team dynamics, or software process characteristics, can improve software defect prediction even more. Future studies can look at the impact of context on defect prediction and create context-sensitive algorithms that can accommodate various software development environments.

Cross Project Defect Prediction: Utilizing information from many software projects to improve defect prediction performance is known as cross-project defect prediction. In light of the heterogeneity of data and the difficulties posed by various software domains and settings, future research can examine efficient approaches for sharing knowledge across projects.

Integration with Software Development Processes: Integration with Software Development Processes: For practical application, defect prediction must be integrated into software development processes. The creation of workflows, tools, and methodologies for seamless integration that allow defect prediction models to be integrated into current development environments can be the focus of future study. This will enable early defect detection and remediation.

Real-Time Defect Prediction: Real-Time defect prediction seeks to enable ongoing monitoring and prediction of problems during software development. The proactive management and prevention of defects can be enabled through future research into approaches for real-time data collecting, processing, and prediction.

Benchmarking and Reproducibility: Securing benchmark datasets, standardized evaluation procedures, and the ability to reproduce findings are significant obstacles in defect prediction research. In the future, efforts can be directed towards developing standard benchmark datasets and evaluation criteria, encouraging the use of open-source software, and allowing the comparison of results from various studies.

In order to progress the field of software defect prediction, it is important to address these future directions and research problems. Doing so will result in models that are more precise, comprehensible, and context-aware, which will make it easier to create software systems that are of high quality.

Conclusion: This paper focuses on software defect prediction using machine learning techniques as a strategy to enhance software quality and reduce development costs. It highlights the need of proactive defect management and draws attention to the shortcomings of conventional methods. In addition to presenting assessment criteria and experimental design, the research investigates various machine learning methods for defect prediction. It contrasts the algorithms and examines the way forward, incorporating cutting-edge techniques and enhancing interpretability, among other things. In conclusion, this paper provides insightful information on software fault prediction using machine learning approaches. It provides useful insights into the theoretical underpinnings, practical considerations, and future directions of this significant research topic. Product development teams can proactively identify and address probable flaws by using machine learning approaches for defect prediction, which will increase the quality of the product and customer satisfaction.

REFERENCES

[1] Arora, I.; Tetarwal, V.; Saha (2015): A. Open issues in software defect prediction. Procedia Comput. Sci., 46, 906-912.

[2] I. C. Society, "IEEE 729-1983 - IEEE Standard Glossary of Software Engineering Terminology," 1982.

[3] W. Bi, (2013). "Research on Software Defect Classification and Analysis," Computer Science.

[4] Grishma, B.R.; Anjali, C. (2015). Software root cause prediction using clustering techniques: A review. In Proceedings of the 2015 Global Conference on Communication Technologies (GCCT) IEEE, Red Hook, NY, USA, 23-24 April 2015; pp. 511–515.

[5] L. Macaulay, Human-computer interaction for software designers, Itp-Media, 1995.

[6] Grishma, B.R.; Anjali, C. Software root cause prediction using clustering techniques: A review. In Proceedings of the 2015 Global Conference on Communication Technologies (GCCT) IEEE, Red Hook, NY, USA, 23–24 April 2015; pp. 511–515

[7] I. Raphael and C. Michael, "Fault links: identifying module and fault types and their relationship," 2004

[8] L. Meng-ren, "Research on Software Defects Classification," Application Research of Computers, 2004.

[9] Meiliana, S. Karim, H. L. H. S. Warnars, F. L. Gaol, E. Abdul rahman, and B. Soewito, (2017). Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset, in 2017,IEEE InternationalConference on Cybernetics and Computational Intelligence (CyberneticsCom), 2017, pp. 19–23. doi:10.1109/CYBERNETICSCOM.2017. 8311708

[10] C. Catal and B. Diri. A systematic review of software fault prediction studies, Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009. doi: 10.1016/j.eswa.2008.10.027 .