Journal of Theoretical and Applied Information Technology
15th May 2023. Vol.101. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# PREDICTING SQL QUERY QUALITY USING MACHINE LEARNING TECHNIQUES

**MOHAMMED RADI**

Department Of Computer Science, Alaqsa University, Gaza,Palestine

E-mail: moh_radi@alaqsa.edu.ps

**ABSTRACT**

To achieve high database performance (e.g., high throughput and low latency), a database tuning technique is needed to make a database application run faster and respond to end-users on time. Users rely heavily on Structure Query Language (SQL) queries to manage and manipulate their data. The complexity of these queries can range from very simple to very complex. Poorly constructed queries usually lead to performance problems. However, the end-user does not know if the SQL statement is poorly written resulting in poor database system performance. Therefore evaluating SQL queries can be difficult because there are many syntactic structures for equivalent queries. Manual evaluation is far too time-consuming since there are so many queries in question. Several papers have provided hints and tips on writing good SQL queries to achieve better performance. However, there is a lack of research on identifying poorly-written SQL queries. Therefore, new approaches are needed to automatically identify poorly-written SQL queries, which have to rewritten for faster performance. In this paper, we propose a classification framework to automatically identify well and poorly-written SQL queries. The proposed framework utilizes various machine learning algorithms including Decision Trees, k Nearest Neighbours, Support Vector Machine, and Naive Bayes. In addition, we identified the key features using two different feature extraction techniques namely TFIDF and Count Vectorizer. To effectively evaluate the proposed framework, we used the Delphi technique to manually label two different datasets namely (Bombay and ERPNext). The experimental results demonstrate that the four machine learning classifiers capable to classify the SQL queries into (well, accepted, and poorly) provide promising results in terms of Recall, precision, and F1-score. In both datasets, the Decision Trees classifier outperform other classifiers by achieving (90%) on the Bombay Dataset and (84%) on the ERPNext Dataset in term of F1-measure. Furthermore, the Count Vectorizer outperforms the TFIDF in predicting poorly written queries.Additionally, the proposed framework can serve as a useful tool for database developers and SQL programmers for detecting poorly written query, consequently utilized for optimizing SQL query performance.

**Keywords:** *SQL Query, Machine Learning, Classification, Feature Selection, Database Systems*

## 1. INTRODUCTION

Database management systems (DBMS) are designed to provide timely answers to end-users. The basic unit of interaction with end users in SQL queries. End users often expect their queries to return results as quickly as possible[1]. However, many factors can affect the response time of the database systems. One of the famous factors is poorly structured or poorly written SQL queries. Most of the current database systems perform an automatic optimization for end-users queries. Even though, still, the well-written SQL queries outperform poorly written ones[2]. For better performance, we need to use faster and more efficient queries.

The purpose of performance tuning in a database management system is to maximize throughput while minimizing response times, which means using the least number of resources while providing the minimum response times. A database management system's performance tuning helps minimize response time and achieve optimal cost by optimizing performance. With minimal response times and maximum throughput, this ensures efficient resource utilization. In the same way, The process of tuning SQL queries is rewriting them so that they are more efficient [3]. However, end-users are unaware that the SQL statement is poorly written and will result in poor performance, as well as the reason why this happens.

As there are several syntactic structures for the same query, assessing SQL queries is not an easy task. In addition, the manual assessment would be far too time-consuming given the number of possible syntactical structures of SQL queries. Therefore, an automatic classification of SQL queries is an important approach to the evaluation of SQL queries. Several works have discussed and introduced some important hints and tips for writing SQL to get better performance[2, 4-8] , these hints are summarized in table 1.

*Table1: Summarization of SQL hints*

| No | Hint |
|---|---|
| 1. | Use Column Names Instead of * in a SELECT Statement. |
| 2. | Avoid including a HAVING clause in SELECT statements. |
| 3. | Avoid using unnecessary DISTINCT Conditions. |
| 4. | Use WHERE instead of HAVING in Join. |
| 5. | Create joins with INNER JOIN (not WHERE). |
| 6. | Avoid nest sub queries. |
| 7. | Use EXISTS instead of DISTINCT. |
| 8. | Try to Use UNION ALL in place of UNION. |
| 9. | Avoid using OR in join conditions. |
| 10. | Avoid functions on the right-hand side of the operator. |
| 11. | Remove any redundant mathematics. |

There has been limited research on the identification of well-written and poorly-written Structured Query Language (SQL) queries. Table 1 suggests various hints and tips that can be used to evaluate the quality of an SQL query, based on the extent to which these hints are followed in the written query. A well-written query is defined as one that follows a large number of hints and tips, while an accepted query is defined as one that follows a neutral number of hints and tips. A poorly-written query is defined as one that follows a small number of hints and tips. However, there is currently a lack of research on identifying both well-written and poorly-written SQL queries.

This paper aims to introduce framework to identify well-written, accepted, and poorly-written SQL queriesusing machine learning classification. The proposed framework consists of four main phases: manual labeling, preprocessing, training, prediction, and evaluation. The Delphi technique is utilized for manual labeling, and two different feature extraction techniques (TFIDF and Count Vectorizer) are employed. The classification step includes the comparison of four machine learning algorithms: Decision Trees, k Nearest Neighbors, Support Vector Machines, and Naive Bayes. The experimental results on two datasets (Bombay and ERPNext) demonstrate that these four machine learning classifiers are capable of classifying SQL queries into three categories (well-written, accepted, and poorly-written) and provide promising results in terms of recall, precision, and F1-score. The proposed framework has the potential to be valuable for researchers, database developers, and SQL programmers in automatically evaluating SQL queries and detecting poorly written ones. It may also be an essential component of comprehensive tools for optimizing SQL query performance. This could improve database performance and have a positive impact on the field of database tuning.

This paper is structured as follows: In Section 2, the related works are presented. The methodology is described in Section 3. The experimental design and datasets used in the study are discussed in Section 4. The findings and discussion are presented in Section 5. The conclusion, which summarizes the key findings of the study and suggests potential avenues for future research, is provided in Section 6.

## 2. RELATED WORKS

Junior SQL developers and recent IT graduates often lack experience in writing effective Structured Query Language (SQL) queries. To reduce the costs associated with queries (including time, space, and complexity), these individuals need to be able to write queries that are less expensive or well-structured. Several methods can be used to find an optimized query, such as hit-and-trial, but developers may still produce queries that are costly to execute. This highlights the need for more effective approaches for identifying and improving poorly written queries [9].

In a previous study[9], the authors proposed the Local Engine for SQL Developer (SLED) as a tool for training junior developers in writing efficient Structured Query Language (SQL) queries. The SLED recommends frequently used or less expensive queries to the developer. As previously mentioned, SQL tuning involves rewriting poorly written queries to improve their performance. The optimization of SQL queries has long been a topic of interest in the field of database research [9]. SQL queries can be written in various forms that produce the same result, but with differing costs and performance [10].

One effective way to optimize query performance is to write the queries in different structures and compare their read and execution plans [4]. There are various techniques that can be used to try to optimize database queries[1-5, 11]. Adhering to general tips for improving SQL queries will generally lead to improved performance of SQL statements.

To improve the performance of Structured Query Language (SQL) queries, various studies [1, 2, 5, 12]have applied various query tuning tips and tricks and presented sets of rules and techniques for optimizing and rewriting poorly written queries. These studies have used experimental methods to demonstrate the effectiveness of these approaches in improving SQL performance. Previous research [8, 13] has utilized machine learning techniques to automatically identify both well-written and poorly-written code, analyzing students' input to programming tasks through static code analysis and using three classification methods (K Nearest Neighbors, Naive Bayes, and Decision Trees). Based on the static analysis and available training data, 21 features were selected for classification, and the performance of the classifiers was evaluated using recall and precision metrics. The results showed that the classifiers were better able to recognize well-written code than poorly-written code, potentially due to the larger amount of well-written code in the training data.

Other studies have also applied machine learning techniques to analyze source code and improve software intelligence [13, 14]. The authors [13] have studied the performance of six classification methods, notably the DTree and stochastic gradient descent (SGD) classifiers with thirteen NASA metric data program (MDP) datasets. The study used a set of software metrics as criteria for classifying software. Several derived measures have been used in the study to assess the classification method, including F-measure, recall, precision, accuracy, and the Matthew correlation coefficient (MCC).

The authors [14]tend to analyze the source code using machine learning techniques to increase the intelligence of the software and make the best use of modern architectures. The findings demonstrate that applying machine learning models in the source code to automatically choose the best energy scaling construction is practical and can be used in the context of auto-system configuration for energy minimization. In addition, machine learning was used to classify an error message generated by a Static Code Analysis (SCA) tool as true-positive, false-positive, or false-negative.The study by [15]. study mainly used measurement metrics to compare the performance of four classifiers, more specifically SVM, KNN, Random Forests, and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) over eight datasets using 22 features Selected Software Engineering Metrics.

To the best of our knowledge, this is the first study utilized machine learning algorithms for predicting sql query quality. In this study, the use of several machine learning algorithms is proposed for the automatic identification of well-written and poorly-written SQL queries. The algorithms employed in this study include k-nearest neighbors, decision trees, support vector machines, and Naive Bayes. Additionally, the key features that can be used to determine the quality of these queries are identified.

## 3. METHODOLOGY

This section is introducing the components of the proposed framework for classifying SQL queries. The framework consists of four main components: manual query labeling, query pre-processing (parsing), model training, and prediction & evaluation. The steps involved in each of these components are illustrated in Figure 1.

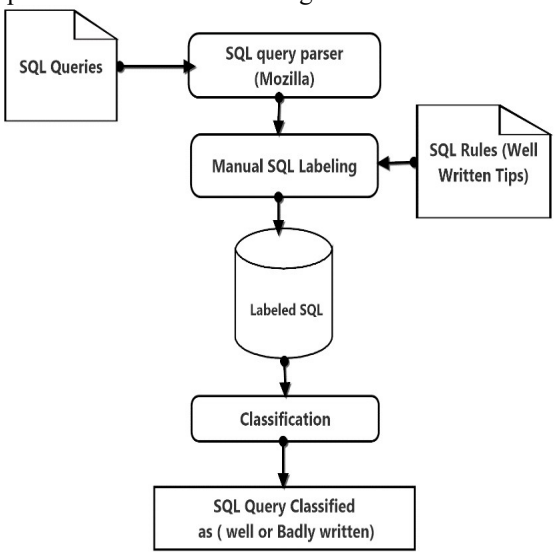


*Figure 1: Query classification Framework overview.*

### 3.1 Manual Labeling Phase

To obtain labeled data for the supervised learning approach, we used the Delphi technique to manually classify the SQL queries into two classes: Well and Poor. The Poor queries were then further divided

into two subclasses: Poor and Accepted. The Delphi technique [16]is a method for structuring group communication, capturing judgments from areas with incomplete research evidence, and resolving differences of opinion. The group for this study included five university lecturers, database developers, and database administrators.

In the first round, the SQL queries from the two datasets were sent to the experts, who were asked to classify them into the given classes. The data from the first round was collected, analyzed, and reduced by determining the SQL queries that received a common class from the majority of the 15 experts. The final label for these determined SQL queries was then determined and they were excluded from the next round.

In the second round, the remaining SQL queries were sent to the same experts with statistical information about the responses from the first round. The experts were then asked to reclassify each query based on the given statistical information. The data from the second round was collected, and the common class for each SQL query was considered as the final label.

For the third-round questionnaire, statistical summaries, medians, and ranges of responses from the previous questionnaire were provided for the SQL queries. The experts were again asked to rate the final label (from the first and second rounds) on a 5-point Likert scale. The SQL queries with an average label of less than 3 were excluded from the dataset. Table 2 presents a sample of the poorly written queries resulting from the manual labeling using the Delphi technique concerning the violated hints presented in Table 1.

*Table 2: Sample Of The Badly Written Queries*

| No | Query | Violated hints |
|---|---|---|
| 1. | select id, name,time_slot_id, semester, a year from student natural join takes natural join section) select id, name from try group by id, name,time_slot_id, semester, year having count(*)>1; | 2,4,6, |
| 2. | select t1.account, sum(t1.debit) - sum(t1.credit) as balance, sum(t1.debit_in_account_currency)- m(t1.credit_in_account_currency) as balance_in_account_currency from `tabGL Entry` t1, `tabAccount` t2 where t1.account = t2.name and t2.root_type = 'Expense' and t2.docstatus < 2 and t2.company = '_Test Company' and t1.posting_date between '2017-01-01' and '2017-02-09' group by t1.account having sum(t1.debit) > sum(t1.credit) limit 1; | 2,4,10 |
| 3. | select distinct bom_item.parent from `tabBOM Item` bom_item where bom_item.bom_no = 'BOM-_Test FG Item 2-001' and bom_item.docstatus = 1 and exists (select * from `tabBOM` where name = bom_item.parent and docstatus = 1 and is_active = 1; | 3,6,7 |
| 4. | select course_id,count(distinct id) from takes group by course_id union select course_id, credits from course where course_id not in (select course_id from takes); | 3,6,7,8 |
| 5. | select distinct course_id,title from section natural join course where semester='spring' and year=2010 and course_id not in (select distinct course_id from prereq); | 3,5,6,7 |
| 6. | select * from (select distinct course_id from teaches where year=2010) arbit1 join (select distinct course_id from prereq) arbit2 on arbit1.course_id=arbit2.course_id; | 1,3,6,7 |
| 7. | select id, name from student where id in (select distinct s.id from (select * from takes natural join section ) s,(select * from takes natural join section ) t where s.id=t.id and s.time _slot_id = t.time_slot_id and s.course_id <> t.course_id); | 1,3,5,6,7 |

### 3.2 Pre-Processing Phase

Data pre-processing is an essential step in any data mining procedure, as it can improve classification results in some cases [6]. This step includes input data cleaning, which involves eliminating SQL queries that contain syntactical errors. To identify such errors, the Mozilla query parser is used. The second step is tokenization, which involves breaking up SQL statements into words, symbols, or other elements called tokens. This process is important because it helps to extract the relevant features from the SQL queries.

### 3.3 Feature Extraction

The purpose of the feature selection step is to obtain the most informative set of features that can be used to enhance the classifier efficiency. Since Each SQL query is a list of words, in this case determining which terms in the query are most distinguishing for that query can be considered an informative feature. However, machines are unable to process text data in its raw form. To make the text understandable to machines, it must first be converted into a format that can be easily interpreted by computers (this is what Natural Language Processing does). Term Frequency-Inverse Document Frequency (TF-IDF) and Bag-of-Words (Bow) are two popular methods for converting text sentences into numerical vectors. For extracting query features, both TF-IDF and Bow have been utilized. The TF-IDF consists of two parts. First TF estimates how important a term or token is in a query. The more occurrence of a term in a query the more important it is. The TF is just as you do for the count vector of term occurrences in a query. (Formally TF defined in equation 1) First, the frequency $f_{t_k,q}$ of each term occurring in the query is counted. Note that if the term $t_k$ does not appear in the text of the query, then $f_{t_k,q}$ is zero. The term frequency of each term $t_k$ of each query q is computed as such:

$$TF_{t_k,q} = \frac{f_{t_k,q}}{maxf_q} \quad (1)$$

where $maxf_q$ indicates the maximum term frequency of all terms that appear in the query. The second part is that IDF is calculated by taking the total number of queries in the corpus and dividing it by the number of queries where the term appears. (Formally IDF defined in equation 2)

$$IDF_t = log\frac{|Q|}{|Q_t|} \quad (2)$$

Where $|Q|$ is the total number of queries, and $|Q_t|$ is the number of queries where the term t appears. By using the IDF of the formula, terms with greater importance get more weight and terms with less importance get fewer weight terms with more importance are given more weight, and terms with less importance are given less weight. The result of this step is a vector of integers presenting the TF-IDF of each query. Comparatively, the bag-of-words approach (BoW) also well-known as Count Vectorizer considers each word count as a feature. Bags-of-words are representations of text that illustrate the occurrence of words within a text. In this approach, two things are involved: first, a vocabulary of known words, and second, a measure of the count of their presence. During this process, any information about the order or structure of words in the text is discarded. It only cares if known words appear in the text, not where they appear.

### 3.4 Classification

The task of predicting a class label for input is essentially a classification problem. Part of this research is to divide the SQL class into three categories: well-written queries, accepted queries, and poorly written queries. According to this model, unseen SQL is classified into three predefined categories based on the predictions of the model. The classification process usually consists of two phases the training phase and the prediction phase. In the training phase, the data is decomposed into a set of features based on feature generation models, such as the vector space model for text data. The next subsection presents the feature extraction method used in this study. As stated earlier, the classification approach is regarded as a modeling problem where a class label is predicated on a given example of input data. Class labels usually string values, in our case. “well-written” “poor-written,” and “accepted”. These ordinal data labels have to be encoded to a unique ordinal integer value, e.g., “well-written” =2, “accepted “=1, and “poor-written” =0 before being provided to the classifier. In a situation of classifying a set of SQL queries into three groups, a binary classifier is needed for this task. The classifier is processed to predict a discrete probability distribution of an example belonging to a specific class. In this study, the most popular machine learning classifiers include Logistic Regression, k Nearest Neighbours, Decision Trees, Support Vector Machine, and Naive Bayes[11]. are separately utilized in the proposed query classification Framework.

### 3.5 Evaluation

The well-known standard evaluation metrics Precision, Recall, and F-score are used on the test set to report the performance of each classifier on the dataset. The detailed calculation of the used matrices is given as follows:

$$Precision = \frac{TP}{TP+F} \quad (3)$$

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

$$F1 - score = 2 \times \frac{Recall \times Precision}{Recall + precisio} \quad (5)$$

Where P is the Positive predicted value, N is the Negative predicted value, TP stands for the True Positive predicted value, TN is the True Negative predicted value, FP stands for the False Positive predicted value and FN is the False Negative predicted value.

## 4. EXPERIMENTAL RESULTS AND DISCUSSIONS

This set of experiments aims to evaluate the classification performance of various machine learning techniques in categorizing SQL queries as well-written or poorly-written and to identify the most suitable feature for classification. The results of these experiments will provide insight into the effectiveness of different machine learning algorithms and feature extraction techniques in this task.

### 4.1 Dataset

In this study two commonly used datasets in query research issues are chosen, which are Bombay [17] and ERPNext [18]. The Bombay dataset consists of SQL queries from students enrolled in an undergraduate database course offered by IIT Bombay from 2015 to 2017. The ERPNext queries, on the other hand, are from the ERPNext software system, an end-to-end business information management solution. The queries were collected by running the test suites of the ERPNext system and evaluating the generated database logs.

To improve the quality of the two datasets, the cleansing process removes queries with syntactical errors that do not pass the SQL parsing process, as well as duplicate queries.

Table 3 shows a statistical summary of the individual record queries and the number of queries selected in our experiments. While table4 and table 5 show the statistical summary of the output of the manual labeling process.

### 4.2 Experiment Setting

To evaluate the performance of the machine learning classifiers, a well-established backtesting method called k-fold cross-validation [7] The first step is to apply k-fold cross-validation (with k=5) by dividing the dataset into subsets. Each dataset (subset) is divided into two subsets: Training and Testing. The training subsets are organized by randomly selecting 80% of the dataset and 20% is considered the testing subset. Hence, the training set was used to fit the classification model, while the testing set was used to evaluate the fit of the model

*Table 3. Datasets statistics summarization*

| Dataset | Total number of queries | Number of parsable queries | Number of distinct query strings | Correct query |
|---|---|---|---|---|
| IIT Bombay | 982 | 934 | 629 | 479 |
| ERPNext | 18,454 | 17,761 | 1,631 | 1,517 |

The experiments were conducted in two sets: one set for two class labels (well and poor) and another set for three class labels (well, accepted, and poor). Each set of experiments was run on two datasets. Four different machine learning classifiers were compared, and the effectiveness of each classifier was assessed based on the comparison of their accuracy using precision, recall, and F1-score metrics. The experiments were conducted using two different features: the first set used TFIDF features and the second set used count vector Bow features. Detailed experimental results are presented in the following section.

*Table 4. Binary manual labeling Datasets statistics summarization*

| Dataset | Correct query | Well | Poor |
|---|---|---|---|
| IIT Bombay | 479 | 120 | 359 |
| ERPNext | 1,517 | 1,127 | 390 |

*Table 5. Ternary manual labeling Datasets statistics summarization*

| Dataset | Correct query | Well | accepted | Poor |
|---|---|---|---|---|
| IIT Bombay | 479 | 120 | 182 | 177 |
| ERPNext | 1,517 | 1127 | 326 | 64 |

### 4.3 Results And Discussions

This section presents the experimental results in terms of classification accuracy using the evolution metrics Recall, Precision, and F1-Measure. For the classification of the experimental datasets, the most popular machine learning classifiers were trained, including Decision Trees (DTC), k Nearest Neighbours (KNN), Support Vector Machine (SVM), and Naive Bayes (NB).

The classification results using two class labels (Poor, Well)for the ERPNext query dataset are shown in Table 6 and the results for the Bombay dataset are shown in Table 7. Sorting the performance of the classifiers with TFIDF and Bow count vectorizer features in terms of recall, DTC ranks first in classifying queries into poorly written and well-written categories and SVM is better than the others. However, looking at the accuracy metrics for each category, this is not necessarily the case. Since the F-score measures the balance of precision and recall, a higher value of the F-score means better performance. Therefore, the weighted accuracy is considered by taking the average of all classes in that case ("Poor", "Well") of the proportion of correct predictions in that class (i.e., the number of correctly predicted instances in that class divided by the total number of objects in that class). When considering the weighting accuracy in terms of F-score, both DTC and SVM achieved the best result in each case.

*Table 6. ERPNext dataset results.*

| | | TFIDF | | CountVectorizer | |
|---|---|---|---|---|---|
| | | Poor | Well | Poor | Well |
| DTC | precision | 0.76 | 0.92 | 0.74 | 0.92 |
| | Recall | 0.76 | 0.92 | 0.77 | 0.91 |
| | F1-score | 0.76 | 0.92 | 0.76 | 0.91 |
| KNN | precision | 0.67 | 0.83 | 0.75 | 0.89 |
| | recall | 0.44 | 0.93 | 0.67 | 0.92 |
| | F1-score | 0.53 | 0.87 | 0.71 | 0.91 |
| SVM | precision | 0.96 | 0.86 | 0.79 | 0.91 |
| | recall | 0.55 | 0.99 | 0.72 | 0.93 |
| | F1-score | 0.70 | 0.92 | 0.75 | 0.92 |
| NB | precision | 0.38 | 0.86 | 0.39 | 0.87 |
| | recall | 0.73 | 0.59 | 0.75 | 0.59 |
| | F1-score | 0.50 | 0.70 | 0.51 | 0.70 |

Consequently, Table 7 presents the result achieved by each classifier in the Bombay queries dataset. The results are somehow similar to the ERPNext dataset, the result shows that the DTC, and SVM achieved the best performance and KNN is better than NB.

As shown in table 6, and table 7 the 4 machine learning classifiers provide a hopeful result in identifying well, accepted, and poorly written SQL queries in terms of recall, precision, and F1 score. However, in the case of using TFIDF as numerical features for modeling the SQL query code better accuracy is achieved in predicting well and accepted written SQL queries. In contrast, the count vectorizer feature shows better performance when predicting Poorly-written ones. In conclusion, the experiments show that using machine learning models to automatically classify SQL queries is feasible and has the potential to be used in the context of comprehensive tools to optimize SQL query performance.

*Table 7. Bombay dataset results.*

| | | TFIDF | | CountVectorizer | |
|---|---|---|---|---|---|
| | | Poor | Well | Poor | Well |
| DTC | precision | 0.97 | 0.90 | 1 | 0.80 |
| | Recall | 0.97 | 0.93 | 0.92 | 0.99 |
| | F1-score | 0.97 | 0.91 | 0.96 | 0.89 |
| KNN | precision | 0.92 | 0.84 | 0.75 | 0.26 |
| | recall | 0.95 | 0.74 | 0.63 | 0.93 |
| | F1-score | 0.93 | 0.79 | 0.68 | 0.31 |
| SVM | precision | 0.90 | 0.89 | 1 | 0.85 |
| | recall | 0.97 | 0.68 | 0.94 | 0.99 |
| | F1-score | 0.93 | 0.77 | 0.97 | 0.92 |
| NB | precision | 0.98 | 0.83 | 0.95 | 0.83 |
| | recall | 0.94 | 0.93 | 0.94 | 0.84 |
| | F1-score | 0.96 | 0.88 | 0.94 | 0.84 |

The classification results using the three-class label for the ERPNext query dataset are shown in Table 8 and the results for the Bombay dataset are shown in Table 9.

DTC ranks first in classifying queries into poor, Accepted, and well-written categories followed by SVM, KNN, and NB. For the ERPNext dataset, DTC apple detects the poorly written query more accurately than other classifiers where it achieves 0.84 and 0.77 F1-score using CountVectorizer and TFIDF respectively. However, for the Bombay dataset DTC apple detects the poorly written query more accurately than other classifiers where it achieves a 0.9 F1-score using CountVectorizer. Moreover, DTC and SVM perform better than others with a 0.86 F1-score using TFIDF.

*Table 8. ERPNext dataset results.*

| | | TFIDF | | | CountVectorizer | | |
|---|---|---|---|---|---|---|---|
| | | Poor | Accepted | Well | Poor | Accepted | Well |
| DTC | Precision | 0.74 | 0.76 | 0.92 | 0.82 | 0.72 | 0.92 |
| | Recall | 0.81 | 0.72 | 0.93 | 0.86 | 0.71 | 0.92 |
| | F1-score | 0.77 | 0.74 | 0.92 | 0.84 | 0.71 | 0.92 |
| KNN | Precision | 0.65 | 0.64 | 0.82 | 0.9 | 0.72 | 0.89 |
| | Recall | 0.57 | 0.38 | 0.92 | 0.59 | 0.67 | 0.92 |
| | F1-score | 0.61 | 0.48 | 0.87 | 0.72 | 0.7 | 0.91 |
| SVM | Precision | 0.9 | 0.98 | 0.86 | 0.83 | 0.75 | 0.91 |
| | Recall | 0.56 | 0.52 | 0.99 | 0.81 | 0.68 | 0.94 |
| | F1-score | 0.69 | 0.68 | 0.92 | 0.82 | 0.72 | 0.92 |
| NB | Precision | 0.21 | 0.38 | 0.85 | 0.22 | 0.38 | 0.86 |
| | Recall | 0.56 | 0.6 | 0.63 | 0.59 | 0.6 | 0.63 |
| | F1-score | 0.31 | 0.47 | 0.73 | 0.32 | 0.46 | 0.73 |

*Table 9. Bombay dataset results*

| | | TFIDF | | | CountVectorizer | | |
|---|---|---|---|---|---|---|---|
| | | Poor | Accepted | Well | Poor | Accepted | Well |
| DTC | Precision | 0.86 | 0.84 | 0.92 | 0.88 | 0.9 | 0.86 |
| | Recall | 0.86 | 0.86 | 0.9 | 0.92 | 0.8 | 0.96 |
| | F1-score | 0.86 | 0.85 | 0.91 | 0.9 | 0.85 | 0.91 |
| KNN | Precision | 0.8 | 0.73 | 0.83 | 0.7 | 0.5 | 0.26 |
| | Recall | 0.86 | 0.72 | 0.75 | 0.33 | 0.59 | 0.39 |
| | F1-score | 0.83 | 0.73 | 0.79 | 0.45 | 0.54 | 0.31 |
| SVM | Precision | 0.81 | 0.76 | 0.89 | 0.87 | 0.88 | 0.87 |
| | Recall | 0.92 | 0.76 | 0.72 | 0.89 | 0.78 | 0.99 |
| | F1-score | 0.86 | 0.76 | 0.8 | 0.88 | 0.83 | 0.93 |
| NB | Precision | 0.94 | 0.61 | 0.84 | 0.97 | 0.62 | 0.8 |
| | Recall | 0.36 | 0.92 | 0.96 | 0.5 | 0.88 | 0.86 |
| | F1-score | 0.52 | 0.73 | 0.9 | 0.66 | 0.73 | 0.83 |

The results of the experiments indicate that the decision tree classifier (DTC) with CountVectorizer features performs better than the other classifiers in both two-class and three-class cases for both datasets.

The obtained promising result indicate that the proposed framework is effectively able to detect poorly written SQL query. These promising results of the proposed farmwork encourage the researchers, database developers, and SQL programmers to use the farmwork to detect the poorly written SQL query. Moreover, the proposed farmwork could be an important component of complete tool for optimizing the performance of SQL query.

## 5. CONCLUSION

The management and manipulation of data in a DMS rely heavily on SQL queries. However, poorly constructed SQL queries cause performance problems. In this study, we proposed a framwork for evaluating SQL query quality using machine learning. In this sudy, the effectiveness of various machine learning classifiers in the classification of SQL queries into three categories: well-written, accepted, and poorly written was investigated. The most popular machine learning classifiers used included k- Nearest Neighbours, Decision Trees, Support Vector Machine, and Naive Bayes. different features, such as term frequency, inverse document frequency, and bag-of-words count vector, were utilized in the training and evaluation of the classifiers. The results of the experiments, conducted on two different datasets, demonstrated that the Decision Trees and Support Vector Machine classifiers showed promising results in the classification of SQL queries based on recall, precision, and F1-score. In comparison, the Naive Bayes and k Nearest Neighbours classifiers did not perform as well. Moreover, the overall result shows that the bag-of-words count vectorizer performs better in predicting poorly written queries. This finding is significant as it can aid in the development of more efficient and effective databases by allowing developers to identify and improve upon poorly written queries that may negatively impact performance. Future research could focus on exploring different features of SQL queries to improve classification accuracy. Also may test the proposed framwork with wide range of data set. It would also be beneficial to investigate the use of other classification techniques, such as Deep Reinforcement Learning, in the prediction of the quality of SQL queries. Finally, The proposed framework has the potential to serve as a valuable tool for researchers, database developers, and SQL programmers, as it offers an automated means of evaluating SQL queries and detecting poorly written ones. The framework's application as an essential component of comprehensive tools for optimizing SQL query performance is also noteworthy, as it has the potential to significantly enhance database performance. By improving the efficiency of database tuning, the framework could have a positive impact on the field of database optimization, thereby contributing to the advancement of the domain.

## REFERENCES

[1] S. J. Kamatkar, A. Kamble, A. Viloria, L. Hernández-Fernandez, and E. G. Cali, "Database performance tuning and query optimization." pp. 3-11.

[2] Y. Kornaga, Y. Bazaka, and E. Marienko, "Ways to optimize SQL queries to improve database performance in high–load systems," *Adaptive systems of automatic control,* vol. 2, no. 37, pp. 26-30, 2020.

[3] H. A. Idhaim, "Selecting and tuning the optimal query form of different SQL commands," *International Journal of Business Information Systems,* vol. 30, no. 1, pp. 1-12, 2019.

[4] N. Kumari, "SQL server query optimization techniques-tips for writing efficient and faster queries," *International Journal of Scientific and Research Publications,* vol. 2, no. 6, pp. 1-4, 2012.

[5] J. P. T. Habimana, "Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries," *International Journal of Scientific Technology Research,* vol. 4, pp. 22-26, 2015.

[6] A. Tripathy, A. Agrawal, and S. K. Rath, "Classification of sentiment reviews using n-gram machine learning approach," *Expert Systems with Applications,* vol. 57, pp. 117-126, 2016/09/15/, 2016.

[7] D. A. Ameyaw, Q. Deng, and D. Söffker, "How to evaluate classifier performance in the presence of additional effects: A new POD-based approach allowing certification of machine learning approaches," *Machine Learning with Applications,* vol. 7, pp. 100220, 2022/03/15/, 2022.

[8] V. Barstad, M. Goodwin, and T. Gjøsæter, "Predicting source code quality with static analysis and machine learning." *N*orsk IKT-konferanse for forskning og utdanning. 2014.

[9] S. Z. Barbhuiya, B. Kumar, Z. Azim, and Y. J. Singh, "Suggestive Local Engine for SQL Developer: SLED," *ADBU Journal of Engineering Technology,* vol. 4, 2016.

[10] M. Mansoubi, and A. M. Bidgoli, "Enhancing Performance of Database with Improving Query automatically," *International Journal of Scientific and Research Publications (IJSRP),* vol. 6, no. 1, 2016.

[11] R. Kumari, and S. K. Srivastava, "Machine learning: A review on binary classification," *International Journal of Computer Applications,* vol. 160, no. 7, 2017.

[12] P. Karthik, G. T. Reddy, and E. K. Vanan, "Tuning the SQL Query in order to reduce time consumption," *International Journal of Computer Science Issues (IJCSI),* vol. 9, no. 4, pp. 418, 2012.

[13] N. Patel, A. Mehta, P. Prajapati, and J. Biskitwala, "Code Buddy: A Machine Learning-Based Automatic Source Code Quality Reviewing System." pp. 453-462.

[14] E. Parisi, F. Barchi, A. Bartolini, G. Tagliavini, and A. Acquaviva, "Source Code Classification for Energy Efficiency in Parallel Ultra Low-Power Microcontrollers," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 878-883, 2021.

[15] E. A. Alikhashashneh, R. R. Raje, and J. H. Hill, "Using machine learning techniques to classify and predict static code analysis tool warnings." pp. 1-8.

[16] P. L. Williams, and C. Webb, "The Delphi technique: a methodological discussion," *Journal of advanced nursing,* vol. 19, no. 1, pp. 180-186, 1994.

[17] P. Agrawal, B. Chandra, K. V. Emani, N. Garg, and S. Sudarshan, "Test Data Generation for Database Applications." pp. 1621-1624.

[18] J. Castelein, M. Aniche, M. Soltani, A. Panichella, and A. v. Deursen, "Search-based test data generation for SQL queries," in Proceedings of the 40th International Conference on Software Engineering, Gothenburg, Sweden, 2018, pp. 1220–1230.