Machine Learning Based Embedded Code Multi-Label Classification

YU ZHOU\textsuperscript{1,}, SUXIA CUI\textsuperscript{2,}, (Senior Member, IEEE), AND YONGHUI WANG\textsuperscript{2,} (Senior Member, IEEE)

\textsuperscript{1}Department of Electrical and Computer Engineering, Prairie View A&M University, Prairie View, TX 77446, USA
\textsuperscript{2}Department of Computer Science, Prairie View A&M University, Prairie View, TX 77446, USA

Corresponding author: Yonghui Wang (yonghuiw@yahoo.com)

This work was supported in part by the United States NSF under Grant HRD-1411260 and Grant OAC-1827243, and in part by the U.S. Department of Education under Grant P120A180114.

\textbf{ABSTRACT} With the development of Internet of Things (IoT) technology, embedded based electronic devices have penetrated every corner of our daily lives. As the brain of IoT devices, embedded based micro controller unit (MCU) plays an irreplaceable role. The functions of the MCUs are becoming more and more powerful and complicated, which brings huge challenges to embedded programmers. Embedded code, which is highly related to the hardware resources, differs from other popular programming code. The hardware configuration may be a big challenge to the programmers, who may only be good at software development and algorithm design. Online code searching can be time consuming and cannot guarantee an optimal approach. To solve this problem, in this paper, an embedded code classifier, which is designed to help embedded programmers to search for the most efficient code with precise tags, is demonstrated. A high quality embedded code dataset is built. A tag correlated multi-label machine learning model is developed for the embedded code dataset. The experimental results show that the proposed code dataset structure is proved to be more efficient on embedded code classification. The proposed embedded classifier algorithm shows a promising result on embedded code dataset. And it outperforms the traditional machine learning text classification models.

\textbf{INDEX TERMS} Embedded code classifier, multi-label, tag-correlated, text classification.

\section{I. INTRODUCTION}
The Internet of Things (IoT) has experienced an explosive growth and becomes a technical revolution following the Internet. Embedded based electronic devices, as the key components in IoT, have penetrated every corner of our daily lives. In IoT field, the embedded based microcontroller unit (MCU) plays an irreplaceable role because of its advantages of real time processing and low power consumption. The world’s leading MCU manufactures, such as ST, TI, NXP, and etc., are upgrading their microcontrollers at an unprecedented speed. The functions of the MCU are becoming more and more powerful and complicated as well. The increasingly complex function of the MCU undoubtedly brings huge challenges to embedded programmers.

Embedded code, mostly written in assembly and C language, differs from other popular programming code (e.g. Java, Python). It is highly related to the hardware resources. Specifically, embedded code may include function module selection, register configuration, memory allocation and etc. The cumbersome hardware configuration is an extremely big challenge to the programmers, as most of them are good at software development and algorithm design, but may not be good at hardware configuration. In most cases, embedded programmers refer to online resources such as the relevant forums and official websites to search for sample codes. However, many online embedded code recourses are not well organized. In other words, online embedded code searching is not an optimal approach because various hardware selections may lead to different functions even though the software and algorithms are similar. This procedure is usually very time consuming for embedded programmers to figure out. A high quality embedded code with precise tags is what an embedded programmer desires the most [1].

To solve the above mentioned problem, a tag correlated machine learning based classifier is proposed to assist embedded programmers to search for the best matched code segment through more specific tags. First of all, a high quality embedded code dataset was built. The code segments in the dataset were mainly collected from the official websites and Stack
functions, e.g. the assigned register value and the interrupt segment is its hardware related configuration and some fixed consistent and unique feature within one embedded code which cannot be utilized as the feature of the code. The self-defined variables and algorithms are defined at random, and the code algorithms, as clearly shown in Fig. 1. The input data of our design is embedded code. Each collected embedded code has two parts: code description and code comment, and it is written in NLP style to point to text classifier used in NLP. Whereas the embedded code content is analyzed and typical multi-label text classification algorithms are addressed and compared in Sec. II. The proposed dataset structure and refined multi-label algorithm are proposed and illustrated in Sec. III. Experimental results and data analysis are demonstrated in Sec. IV. Sec. V includes the concluding remarks and the description of future works.

II. EMBEDDED CODE STRUCTURE AND TEXT CLASSIFICATION

Before introducing the proposed design, we first present the structure of embedded code composition and its inherent features. Moreover, the vectorization methods (CountVec, TF-IDFVec) and the typical algorithms for multi-class and multi-label text classification methodology, such as Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), One-to-Rest, Classifier Chains, and etc., are explained. The evaluation metrics are addressed and the advantages and limitations of each algorithm are also considered. At last, popular deep learning model TextCNN is briefly introduced.

A. EMBEDDED CODE COMPOSITION

The input data of our design is embedded code. Each collected embedded code has two parts: code description and code content. The code description is composed of code title and code comment, and it is written in NLP style to point out which application this embedded code is used for. It also make the code easier to be understood by other programmers. The processing methodology of code description is equivalent to text classifier used in NLP. Whereas the embedded code content has its own unique format and feature. The typical structure of the embedded code content is composed of hardware declaration and configuration, self-defined variables, and the code algorithms, as clearly shown in Fig. 1. The self-defined variables and algorithms are defined at random, which cannot be utilized as the feature of the code. The consistent and unique feature within one embedded code segment is its hardware related configuration and some fixed functions, e.g. the assigned register value and the interrupt service function. In other words, the only valuable information hidden in an embedded code segment is its hardware information and the specific function, which are the unique features to be extracted.

As mentioned in Sec. I, embedded code is hardware related. A typical feature of embedded code is that it contains the hardware information of the MCU, such as register of I/O, Timer, AD, and etc. In return, the hardware information reflects which function module and which register is used. It is capable to label a code segment with more precise tags according to the hardware information.

B. TEXT CLASSIFICATION

Text classification is a classic supervised learning algorithm used for assigning tags or categories to text document according to its inherent feature in the content. It is one of the fundamental tasks in NLP field with broad applications such as, news labeling, spam detection, and sentiment analysis [2]. As shown in Fig. 2, the input is text or document, and the output is a predicted class \( c_i \in C \), where \( C \) is a fixed set of classes \( C = \{c_1, c_2, \cdots, c_n\} \).

The step 2 of Fig. 2, called text preprocessing or text cleaning, is applied before the classification model to avoid “garbage in, garbage out”. The text cleaning techniques work extremely well so far in practice. For our particular data set, the text cleaning step includes C language decoding, stop words, punctuation, and bad characters removing, lower case modifying, etc. In step 3, the text in the document is vectorized. Some popular approaches, such as CountVec, TF-IDFVec [3], and Word2Vec are the prevalent models in text vectorization. In this design, both CountVec and TF-IDFVec are tested. (1) gives the mathmathics representation of TF-IDF:

\[
TF-IDF(x) = TF(x) \times IDF(x).
\]  

(1)

where \( TF \) is term frequency, and \( IDF \) is the inverse document frequency, which can be illustrated as:

\[
IDF(x) = \log \frac{N}{n(x)}.
\]  

(2)
Here $N$ represents the total number of texts in the corpus, and $N(x)$ represents the total number of texts in the corpus containing words $x$. In case $N(x) = 0$, $IDF$ is optimized as:

$$IDF(x) = \log \frac{N}{N(x) + 1}.$$  \hfill (3)

With the fast development of data science, machine learning is proved to be efficient on large scale text classification tasks. The work principle diagram of machine learning based text classification can be found in Fig. 3. Typical traditional text classification algorithms, like the Naïve Bayes (NB) [5], Logistic Regression (LR) [6], and Support Vector Machine (SVM) [7], achieve good results on a relative small scale text. With the considerable powerful computing hardware progress, the deep learning model, such as Convolutional Neural Networks (CNN) [8], has proved to be effective on large scale text classification. From traditional approach to the deep learning model, various text classification algorithms have been verified in many application scenarios, such as the news classification, spam email filter, and multi-labeling [9]–[11].

According to specific application scenario, multiclass classification (MCC) and Multi-label classification (MLC) are the dominant two categories in text classification area. As for MCC, the classes are mutually exclusive, the predicted document is labeled with only one unique tag out of a set of classes $C = \{c_1, c_2, \ldots, c_m\}$. Whereas, for MLC, the predicted document is assigned one or more tags, and the tags are somehow related.

In this research, a code segment has one or more functions, which means each code segment is tend to be labeled with one or more classes; therefore, the MLC is chosen. Moreover, some MLC algorithms are generated from MCC, like the One-to-Rest algorithm and Binary Relevance algorithm.

1) NB TEXT CLASSIFICATION

NB text classification algorithm originated from Bayes Theorem. The conditional probabilities of occurrence of two events based on the probabilities of occurrence of each individual event [12], [13]. This means that any vector that represents a text contains the probabilities of word’s appearance within the text of a given category, so that the algorithm can compute the likelihood of that word’s belonging to the category [14]. Naïve Bayes contains a family of statistical algorithms one can make use of when conduct text classification. One of the members of NB family is Multinomial Naïve Bayes (MNB) [3]. Its main advantages include that good results could be achieved when the dataset is small (e.g. a couple of thousand tagged samples) and the computational complexity is low. The mathematics foundation can be illustrated in the following equations.

For a document $d$ and a class $c$:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}.$$ \hfill (4)

The best class in NB classification is the most likely or maximum a posteriori (MAP) class $C_{MAP}$, then:

$$C_{MAP} = \arg \max_{c \in C} P(c|d)$$ \hfill (5)

according to Bayes rule:

$$C_{MAP} = \arg \max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$ \hfill (6)

Dropping the denominator:

$$C_{MAP} = \arg \max_{c \in C} P(d|c)P(c)$$ \hfill (7)

Apply the rules to the full document, then

$$C_{MAP} = \arg \max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$ \hfill (8)

2) SVM TEXT CLASSIFICATION

SVM is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it [3], [7], [15]. It can be applied to any kind of data as long as the data can be converted to vector form. This means that in order to leverage the power of SVM text classification, texts have to be transformed into vectors. SVM is a discriminative classifier formally defined by a separating hyperplane. Given a labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which sets boundary of different categories. In two dimensional spaces, this hyperplane is a line dividing a plane into two parts where each class lays in one side. SVM doesn’t need much training data to start providing accurate results. Although it needs more computational resources than Naïve Bayes, SVM can achieve more accurate results [16].

3) LR TEXT CLASSIFICATION

LR text classification is another scheme applies statistics to machine learning based classification problems. It makes a binary predictive analysis using a logistic function. In this LR-based [6] text classification, only a binary variable is modeled by logistic function. Binary logistic regression and multinomial logistic regression are the popular algorithms used in text classification. In scikit-learn library [3], both of them are performed. In large scale text, LR analysis of high-dimensional data poses computational and statistical challenges [17].
4) DEEP LEARNING BASED TEXT CLASSIFICATION
Convolution neural networks (CNN) have been widely applied in image processing area and achieves excellent results. Originally, CNN was not able to be applied for text classification, because the length of text is not fixed. Instead, CNN was only suitable for data with a constant length, such as images, and the convolution kernel is usually fixed at 3 × 3. In 2004, Yoon Kim [8] applied CNN in text classification with a novel approach, which makes the sentences to the same length by padding for short sentences and deleting for longer ones. In [18], the authors proposed character-level convolutional networks for text classification, extended the classification task from word level to character level. Another deep learning based NLP algorithm is BERT [19], the most up-to-date language representation model, which was introduced in 2018 and provides state-of-the-art results on natural language processing tasks.

C. MULTI-LABEL TEXT CLASSIFICATION
Multi-label learning originated from the investigation of text categorization problem, where each document may belong to several predefined topics simultaneously. In multi-label learning, the training set is composed of instances each associated with a set of labels, and the task is to predict the label sets of unseen instances through analyzing training instances with known label sets [20]. Most traditional learning algorithms are developed for single-label classification problems. Therefore a lot of approaches in the literature transform the multi-label problem into multiple single-label problems, so that the existing single-label algorithms such as the One-to-Rest and Binary Relevance algorithms can be applied. However, the drawback of this scheme is that it cannot predict the underlying correlation between the classes.

1) ONE-TO-REST
Traditional two-class and multi-class problems can both be cast into multi-label ones by restricting each instance to have only one label [14]. And obviously multi-label problems are more challenging. An intuitive approach to solving multi-label problem is to decompose it into multiple independent binary classification problems (one per category) [20]. In one-to-rest algorithm, multiple independent classifiers are built for an unseen instance, and the class with maximized confidence will be selected, the NB, SVM, LR could be embedded into the One-to-Rest.

2) BINARY RELEVANCE
The basic idea of this algorithm is to decompose the multi-label learning problem into q independent binary classification problems, where each binary classification problem corresponds to a possible label in the label space [15], [21]. In this case, an ensemble of single-label binary classifiers is trained, one for each class. Each classifier predicts either the membership or the non-membership of one class. The union of all classes that were predicted is taken as the multi-label output. This approach is popular because it is easy to implement, however it also ignores the possible correlations between class labels [14].

3) CLASSIFIER CHAINS
In [22], J. Read, et al. proposed the classifier chains for multi-label solution and proved its advantageous in exploring tags correlation, which overcomes the disadvantage of Binary Relevance. A chain of binary classifiers \( C_0, C_1, \cdots, C_n \) is constructed in series, where a classifier \( C_j \) uses the predictions of all the classifier \( C_j \), where \( j < i \). The number of classifiers built for this approach is equal to the number of categories [14].

4) ML-KNN
In [20], the authors proposed a lazy learning approach to multi-label learning, which is derived from the traditional K-nearest neighbor (KNN) algorithm. Specifically, for each unseen instance, its K nearest neighbors in the training set are identified first. After that, based on statistical information gained from the label sets of these neighboring instances, i.e. the number of neighboring instances belonging to each possible class, MAP principle is utilized to determine the label set for the unseen instance [23].

5) LABEL POWERSET
Label Powerset is a problem transformation approach to multi-label classification that transforms a multi-label problem to a multi-class problem with one multi-class classifier trained on all unique label combinations found in the training data [14]. The method maps each combination to a unique combination ID number, and performs multi-class classification using the classifier as multi-class classifier and combination IDs as classes. This approach does take possible correlations between class labels into account. More commonly this approach is called the label-powerset method, because it considers each member of the power set of labels in the training set as a single label. This method requires at least worst case \( (2^C_j) \) classifiers, and has a high computational complexity. When the number of classes increases, the number of distinct label combinations can grow exponentially. This easily leads to combinatorial explosion and thus computational infeasibility. Furthermore, some label combinations will have very few positive examples.

D. EVALUATION METRICS FOR MULTI-LABEL TEXT CLASSIFICATION
The classification results are measured by the evaluation metrics introduced in [14]. The predicted tags are compared against ground truth tags which were manually acquired from experienced embedded programmers. After TP (true positives), TN (true negatives), FP (false positives), and FN (false negatives) are obtained, accuracy, precision, recall, and F1-Score are calculated as evaluation factors.

The evaluation measures for single-label are usually different from multi-label. Single-label classification uses simple
metrics such as precision, recall, accuracy, and etc. Specifically, in single-label classification, accuracy is calculated as
\[ \frac{1}{N} \sum_{i=1}^{N} I[y(i) = \hat{y}(i)]. \]

In multi-label classification, the prediction of each label plays a role in final evaluation. To measure a multi-class classifier, the average of all classes are taken into consideration. Two methods, micro-averaging and macro-averaging, were introduced in [14]. In micro-averaging, all TPs, TNs, FPs and FNs for each class are averaged accordingly as illustrated in (9) and (10). And the micro-average F\(_1\) score can be simply obtained by the harmonic mean. Macro-averaging is straightforward, as shown in (11) and (12), the average of the precision and recall of the system on different sets are taken. Macro-averaging measures how the system performs overall across different sets of data, while micro-averaging can be of more useful when dataset varies in size.

Micro-averaging Precision:
\[ P_{\text{micro}}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} [TPs(c_i) + FPs(c_i)]} \]  
(9)

Micro-averaging Recall:
\[ R_{\text{micro}}(D) = \frac{\sum_{c_i \in C} TPs(c_i)}{\sum_{c_i \in C} [TPs(c_i) + FNs(c_i)]} \]  
(10)

Macro-averaging Precision:
\[ P_{\text{macro}}(D) = \frac{\sum_{c_i \in C} P(D, c_i)}{|C|} \]  
(11)

Macro-averaging Recall:
\[ R_{\text{macro}}(D) = \frac{\sum_{c_i \in C} R(D, c_i)}{|C|} \]  
(12)

III. PROPOSED EMBEDDED CODE DATASET STRUCTURE AND METHODOLOGY DESIGN

Before any machine learning classification research can be conducted, the dataset has to be thoroughly studied. Based on our experience on embedded system design, a very popular microcontroller family, PIC, is chosen to build our dataset. The proposed methodology can be extended to any microcontroller family. In this section, we present a novel dataset structure and a multi-label model for the code classifier.

A. THE DATASET BUILDING AND PREPROCESSING

For the proposed data set, there are 1000 code segments distributed in 14 categories. There are 4 categories for application scenarios (LED, LCD, Sensor, Motor) and 10 function module categories (Timer, SPI, I2C, UART, PWM, AD/DA, PORTA, PORTB, PORTC, and PORTD). We perform the classification task on machine learning model for our embedded code classifier. As discussed in Sec. II, the embedded code has two parts: code description and code content. The code description summarizes the general features of the code, such as MCU type, application scenario and some function module information, etc. The rough code information could be extracted from the code description. Obviously, the general information is only a rough narration, which is not sufficient to precisely label the code, because more hardware features, such as the I/O register, the complete function module, are hidden in the code content. However, in the code content, the application usually is not included, because the variables are randomly defined by the programmer. Based on this observation, we proposed our own dataset structure. In the proposed code data set, the code description and code content are reorganized and labeled, the tags for each code segment will be more precise. Fig. 4 demonstrates the advantage of our dataset structure.

As clearly shown in Fig. 4, the tags PIC18, LED, and AD could be extracted from the code description. However, the more precise I/O information is not in the code description, which is hidden in the code content. Since PORTD is used as I/O, the PIC18F, AD, PORTD are labeled accordingly to the code content, but the LED is ignored. By considering both parts, in our data set, we reorganize the dataset and label each code segment for code description and code content simultaneously. The advantage of this operation is to get the tags both from the code description and code content, also the relationship between application tags and function tags can be explored. The drawback of the dataset is the unbalancing of the code description and code content, as it can be seen that the amount of text in embedded code content far exceeds the amount of text in code description. The relative small amount of the text in code description will lead to the low classification accuracy compared with the code content. In the following proposed methodology, the relevance of the label is studied to solve the problem of data imbalance in code description and code content.

B. PROPOSED MODEL DESIGN

After obtaining a code database with tags marking each code segment through preprocessing, a unique parallel cascaded auto code classifier is proposed as shown in Fig. 5. Specifically, there are two machine learning channels for our code classifier, one channel is built to process the code description, and the other channel is designed to analyze the code content. The function of the first channel is to extract the rough features, such as the chip name, application scenario, and the rough function. The second channel is designed to extract the hardware feature and generate more precise tags. In this channel, the precise feature, like the chip model, function module and register information will be extracted according to the inherent hardware information in the code segment.

After training, the multi-label classifier models 1 and 2 are applied to the predictive models 1 and 2 respectively to generate the application tags and function module tags.
Afterwards, the union operation is performed and the tags generated from the two channels will be combined to label each code segment. For the multi-label classifier design, the CountVec and TF-IDFVec vectorization methodologies are both tested; 7 different algorithms including the series of One-to-Rest, ML-KNN, Classifier Chains, Binary Relevance, and Label Powerset are tested, which will be further discussed in Sec. IV.

Moreover, in embedded code, there exists a correlation between the application and function modules, as shown in Fig. 6. For example, LED application is timer and PWM related, motor application is PWM related, and sensor application is AD/DA related. Based on this, we do another exploration of the tags correlation from channels 1 and 2. The objective is to discover the relationship between the application scenario and function modules. In other words, this design will tell us which function modules will be mostly used for a specific application, also find which application will be speculated according to the function modules. There will be a compensation value between the correlated features. The correlations between applications and functions will overcome the drawbacks of the unbalancing problem in the proposed dataset and enhance the application accuracy. In the proposed methodology, the correlations of the features are trained, and the correlation score reflects the specific relationship between an application scenario and a specific function module.

The parameter for the application and function compensation in each category is defined as follows:

$$\alpha = \frac{(K_a)^2}{K_f} \quad (13)$$

where $K_a$ is the number of application scenarios, and $K_f$ is the number of function modules. The compensation value to the application from correlated function in each category is:

$$\delta_a = \begin{cases} \frac{\sum_{0}^{K_a} \lambda_a (i)}{K_a} & \text{if } \lambda_a > 0 \\ 0 & \text{if } \lambda_a \leq 0 \end{cases} \quad (15)$$

where $\delta_a$ is the compensation of function category to application category, $\lambda_a$ is the correlation value in each application category. The compensation value to the function from the correlated application in each category is:

$$\delta_f = \begin{cases} \frac{\sum_{0}^{K_f} \lambda_f (i)}{K_f} & \text{if } \lambda_f > 0 \\ 0 & \text{if } \lambda_f \leq 0 \end{cases} \quad (16)$$

where $\delta_f$ is the compensation of application category to function category, $\lambda_f$ is the correlation value in each application category.

The compensation for all the application categories from code description dataset is:

$$\delta_{at} = \sum_{0}^{K_a} \lambda_a (i), \text{ if } \lambda_a > 0 \quad (17)$$

The compensation for all the function categories from code content dataset is:

$$\delta_{ft} = \sum_{0}^{K_f} \lambda_f (i), \text{ if } \lambda_f > 0 \quad (18)$$
IV. EXPERIMENTS RESULTS ANALYSIS

The experiments are carried out step by step on the Jupiter notebook plus the Scikit-Learn algorithm libraries [3]. The outcomes of the experiments are presented in two categories: the performance of the proposed dataset structure and the comparison between the proposed model and the traditional machine learning approaches. The dataset is split into a training set (67%) and a validation set (33%). Figs. 7 and 8 show the characteristics of the dataset.

For the vectorization approach in text classification, TF-IDFVec outperforms the CountVec because the inverse term frequency is considered as shown in (1) and (2). However for embedded code, there is a large number of self-defined variables, which is at random defined and could be seen as inverse words. This large number of inverse words will lead to the bad effect of TF-IDFVec. Therefore, both the CountVec and TF-IDFVec are tested to demonstrate the vectorization influence on the performance. As for the machine learning classifier model, MNB, SVM, and LR utilized in One-to-Rest, together with Classifier Chains, Binary Relevance, MLKNN, and Label Powerset are tested respectively.

A. PERFORMANCE OF THE PROPOSED DATASET STRUCTURE

In this section, the overall accuracy, $F_1$-Score-macro, $F_1$-Score-micro, and $F_1$-Score-weighted of proposed code
dataset (PCD), code description only (CDD), and code content only (CCD) are compared with 14 different algorithms. Tables 1 to 3, as well as Figs. 9 to 11, demonstrate the performance of accuracy, $F_1$-Score-macro, $F_1$-Score-micro, $F_1$-Score-weighted for the 14 algorithms for PCD, CDD, and CCD, respectively. The highest scores are achieved in PCD (Table 1 and Fig. 9) by CountVec + Binary Relevance (SVM as the core), which are 0.724, 0.845, 0.890, and 0.885, respectively. These results show that PCD provides higher performance than both CDD (results in Table 2 and Fig. 10) and CCD (results in Table 3 and Fig. 11). Under the same algorithm, the performance of PCD shows a more promising result than CDD and CCD. Specifically, CDD has the worst performance, which means that the inherent feature in the code description is not sufficient.

The accuracy, $F_1$-Score-macro, $F_1$-Score-micro, $F_1$-Score-weighted scores are the evaluation metrics for the overall performance of the system. However, in most cases, we want to evaluate the performance of a specific category, therefore, the accuracy scores of each category with One-to-Rest series algorithms are investigated with CountVec and TF-IDFVec respectively. In Tables 4 and 5, the experimental results show that, PCD structure can achieve higher performance in most categories except the 2 application categories, Sensor and Motor. These results demonstrate the reorganized dataset structure is more promising in the embedded
code classification. The reason for the relatively low results of the 2 applications is that, in PCD, the number of the words in the code description is much less than that of the words in the code content, which will lead to the weights of the worse performance in the application category. In order to overcome this problem, we propose the tag correlated machine learning
TABLE 3. The accuracy score, $F_1$-Score-macro, $F_1$-Score-micro, $F_1$-Score-weighted of CCD under different vectorization methods and different machine learning models.

| Method               | Accuracy | $F_1$-Score-macro | $F_1$-Score-micro | $F_1$-Score-weighted |
|----------------------|----------|-------------------|------------------|---------------------|
| CountVec + MNB       | 0.233    | 0.691             | 0.711            | 0.783               |
| CountVec + SVM       | 0.645    | 0.773             | 0.832            | 0.841               |
| CountVec + LR        | 0.664    | 0.788             | 0.869            | 0.853               |
| TF-IDF + MNB         | 0.594    | 0.686             | 0.829            | 0.795               |
| TF-IDF + SVM         | 0.694    | 0.775             | 0.877            | 0.852               |
| TF-IDF + LR          | 0.600    | 0.685             | 0.829            | 0.791               |
| CountVec + Binary Relevance | 0.630 | 0.764             | 0.837            | 0.840               |
| CountVec + Classifer Chains | 0.633 | 0.797             | 0.857            | 0.854               |
| CountVec + Label Powerset | 0.630 | 0.781             | 0.848            | 0.841               |
| CountVec + ML-KNN    | 0.364    | 0.781             | 0.848            | 0.841               |
| TF-IDF + Binary Relevance | 0.570 | 0.769             | 0.872            | 0.844               |
| TF-IDF + Classifer Chains | 0.673 | 0.765             | 0.865            | 0.837               |
| TF-IDF + Label Powerset | 0.703 | 0.789             | 0.879            | 0.857               |
| TF-IDF + ML-KNN      | 0.624    | 0.789             | 0.879            | 0.857               |

TABLE 4. The Accuracy Score in each category on the PCD, CDD, and CCD using One-to-Rest (CountVec + MNB, CountVec + SVM, CountVec + LR).

| Category | CountVec + MNB | CountVec + SVM | CountVec + LR |
|----------|----------------|---------------|---------------|
| PCD      |                |               |               |
| CDD      |                |               |               |
| CCD      |                |               |               |
| LED      | 0.870          | 0.773         | 0.861         |
| LCD      | 0.906          | 0.855         | 0.900         |
| Sensor   | 0.861          | 0.921         | 0.829         |
| Motor    | 0.821          | 0.967         | 0.764         |
| Timer    | 0.900          | 0.697         | 0.906         |
| SPI      | 0.964          | 0.870         | 0.967         |
| I2C      | 0.985          | 0.861         | 0.976         |
| UART     | 0.979          | 0.779         | 0.973         |
| PWM      | 0.938          | 0.852         | 0.958         |
| AD/DA    | 0.973          | 0.800         | 0.973         |
| PORTA    | 0.979          | 0.627         | 0.979         |
| PORTB    | 0.976          | 0.639         | 0.976         |
| PORTC    | 0.979          | 0.645         | 0.976         |
| PORTD    | 0.827          | 0.873         | 0.827         |

TABLE 5. The Accuracy Score in each category on the PCD, CDD, and CCD using One-to-Rest (TF-IDFVec + MNB, TF-IDFVec + SVM, TF-IDFVec + LR).

| Category | TF-IDFVec + MNB | TF-IDFVec + SVM | TF-IDFVec + LR |
|----------|----------------|----------------|---------------|
| PCD      |                |               |               |
| CDD      |                |               |               |
| CCD      |                |               |               |
| LED      | 0.888          | 0.876         | 0.903         |
| LCD      | 0.961          | 0.921         | 0.961         |
| Sensor   | 0.945          | 0.945         | 0.945         |
| Motor    | 0.939          | 0.952         | 0.939         |
| Timer    | 0.904          | 0.813         | 0.904         |
| SPI      | 0.979          | 0.918         | 0.979         |
| I2C      | 0.964          | 0.939         | 0.967         |
| UART     | 0.982          | 0.861         | 0.982         |
| PWM      | 0.979          | 0.897         | 0.982         |
| AD/DA    | 0.948          | 0.833         | 0.955         |
| PORTA    | 0.985          | 0.730         | 0.985         |
| PORTB    | 0.991          | 0.742         | 0.991         |
| PORTC    | 0.988          | 0.742         | 0.988         |
| PORTD    | 0.952          | 0.948         | 0.952         |

B. PERFORMANCE OF THE PROPOSED METHODOLOGY

Before demonstrate the performance of the proposed methodology, we first investigate the correlation effect between the application scenarios and function modules to verify the relationship between them. As shown in Fig. 12, also as speculated before, some of the labels are highly correlated. PORTA, PORTB, PORTC, and PORTD have very high correlation scores between each of them, which is mainly due to the IO configuration shares the same registers. Motor control and PWM also have a high correlation score, 0.22, which means that the motor control code is mostly written by PWM.
This kind of relationship can be also applied to LED and Timer, Sensor and I2C/AD. The correlation values between the tags are clearly illustrated in Fig. 12.

With the formulas discussed in Sec. III, the label results of each category, especially the application categories, are compensated with the correlation values. For the performance of the proposed tag correlated machine learning model, we investigate the accuracy enhancement in each category of the proposed dataset using One-to-Rest algorithms including NB, SVM, and LR. The CountVec and TF-IDF are tested in combination with different algorithms separately. As shown in Table 6, under the same vectorization methodology and classifier (MNB, SVM, LR), the accuracy of each correlated category is enhanced, outperforming the traditional methodologies both in application category and function category. In the application scenario of interest, for the LED, Sensor, and Motor application categories, the highest accuracies are improved to 0.937, 0.991, and 1.0 from the highest accuracy scores of traditional machine learning algorithms (0.906, 0.945, 0.964). The accuracy of LCD remains the same, because there are no correlated tags for it. The performance of the proposed tag correlated model namely Co-MNB, Co-SVM, Co-LR overcomes the bad accuracy problems using MNB, SVM, LR, which is caused by the imbalance between code description and code content. Similar performance results for CDD and CCD are tabulated in Tables 7 and 8, respectively.
TABLE 8. The accuracy performance of the proposed model vs. conventional machine learning model using One-to-Rest (CountVec + MNB, CountVec + SVM, CountVec + LR, TF-IDFVec + MNB, TF-IDFVec + SVM, TF-IDFVec + LR) on CCD.

| Category | CountVec + One-to-Rest | TF-IDFVec + One-to-Rest |
|----------|------------------------|-------------------------|
|          | Co-MNB | Co-SVM | Co-LR | MNB | SVM | LR | Co-MNB | Co-SVM | Co-LR | MNB | SVM | LR |
| LED      | 0.892  | 0.913  | 0.949  | 0.861 | 0.882 | 0.918 | 0.934  | 0.945  | 0.946 | 0.903 | 0.912 | 0.915 |
| LCD      | 0.900  | 0.964  | 0.985  | 0.900  | 0.904  | 0.985  | 0.961  | 0.973  | 0.961 | 0.961  | 0.973  | 0.961 |
| Sensor   | 0.872  | 0.961  | 0.985  | 0.827  | 0.918  | 0.939  | 0.991  | 0.988  | 0.991 | 0.945  | 0.942  | 0.945 |
| Motor    | 0.800  | 0.954  | 0.975  | 0.764  | 0.918  | 0.939  | 0.975  | 0.978  | 0.975 | 0.939  | 0.942  | 0.939 |
| Timer    | 0.908  | 0.947  | 0.972  | 0.906  | 0.945  | 0.970  | 0.966  | 0.966  | 0.963 | 0.964  | 0.964  | 0.961 |
| SPI      | 0.967  | 0.988  | 0.991  | 0.967  | 0.988  | 0.991  | 0.979  | 0.991  | 0.979 | 0.991  | 0.991  | 0.979 |
| I2C      | 0.979  | 0.988  | 0.991  | 0.976  | 0.985  | 0.988  | 0.970  | 0.988  | 0.961 | 0.967  | 0.985  | 0.958 |
| UART     | 0.973  | 0.976  | 0.982  | 0.973  | 0.976  | 0.982  | 0.982  | 0.985  | 0.987 | 0.982  | 0.988  | 0.979 |
| PWM      | 0.966  | 0.984  | 0.997  | 0.958  | 0.976  | 0.988  | 0.990  | 0.997  | 0.987 | 0.982  | 0.988  | 0.979 |
| ADDA     | 0.974  | 0.971  | 0.968  | 0.973  | 0.970  | 0.967  | 0.956  | 0.965  | 0.941 | 0.955  | 0.964  | 0.939 |
| PORTA    | 0.979  | 0.986  | 0.979  | 0.979  | 0.985  | 0.979  | 0.983  | 0.989  | 0.989 | 0.988  | 0.988  | 0.988 |
| PORTB    | 0.977  | 0.992  | 0.986  | 0.976  | 0.991  | 0.985  | 0.986  | 0.995  | 0.992 | 0.985  | 0.994  | 0.991 |
| PORTC    | 0.980  | 0.989  | 0.989  | 0.976  | 0.985  | 0.985  | 0.989  | 0.992  | 0.992 | 0.985  | 0.988  | 0.988 |
| PORTD    | 0.830  | 0.985  | 0.981  | 0.827  | 0.982  | 0.979  | 0.954  | 0.988  | 0.966 | 0.952  | 0.985  | 0.964 |

TABLE 9. The accuracy performance of the proposed model vs. conventional machine learning model using different classifier algorithms.

| Proposed Methodology | Accuracy | Conventional Methodology | Accuracy |
|----------------------|----------|--------------------------|----------|
| CO + CountVec + MNB  | 0.385    | CountVec + MNB           | 0.273    |
| CO + CountVec + SVM  | 0.827    | CountVec + SVM           | 0.715    |
| CO + CountVec + LR   | 0.809    | CountVec + LR            | 0.697    |
| CO + TF-IDF + MNB    | 0.706    | TF-IDF + MNB             | 0.594    |
| CO + TF-IDF + SVM    | 0.715    | TF-IDF + SVM             | 0.691    |
| CO + TF-IDF + LR     | 0.615    | TF-IDF + LR              | 0.591    |
| CO + CountVec + Binary Relevance | 0.748 | CountVec + Binary Relevance | 0.724 |
| CO + CountVec + Classifier Chains | 0.745 | CountVec + Classifier chains | 0.721 |
| CO + CountVec + Label Powerset | 0.679 | CountVec + Label Powerset | 0.655 |
| CO + CountVec + ML-KNN | 0.597 | CountVec + ML-KNN        | 0.573    |
| CO + TF-IDF + Binary Relevance | 0.697 | TF-IDF + Binary Relevance | 0.673 |
| CO + TF-IDF + Classifier Chains | 0.706 | TF-IDF + Classifier chains | 0.682 |
| CO + TF-IDF + Label Powerset | 0.736 | TF-IDF + Label Powerset | 0.712 |
| CO + TF-IDF + ML-KNN | 0.633 | TF-IDF + ML-KNN          | 0.609    |

FIGURE 13. The accuracy score of proposed model on PCD, CDD, CCD.

In Table 9, the accuracy scores of the proposed model with 14 different classification algorithms under CountVec and TF-IDFVec are compared for PCD, CDD, CCD dataset, the results show that the accuracy scores of the 3 kinds of...
data under the proposed models (left part) are all enhanced comparing to the traditional machine learning models (right part). For PCD, CDD, and CCD, the best algorithm (CO + CountVec + SVM) achieves the highest accuracy scores, which are 0.827, 0.206, and 0.757. These values are all higher than the corresponding scores of traditional method (0.715, 0.094, 0.645). These comparisons are also clearly illustrated in Figs. 13 and 14.

From all the above results, the proposed tag correlated model plus the proposed dataset are proved to work more efficiently, outperforming the traditional dataset structure and traditional machine learning models.

V. CONCLUSION

In this paper, a machine learning based auto embedded code classifier is proposed to help embedded programmers to better query high quality code segments. Multi-label methodology is utilized to precisely label the code segments. Firstly, a dataset with high quality embedded code segments of PIC MCU is built. The label classes cover 4 application scenarios (LED, LCD, Sensor, Motor) and 10 function modules (Timer, SPI, I2C, UART, PWM, AD/DA, PORTA, PORTB, PORTC, and PORTD). Moreover, a novel dataset structure is designed to overcome the insufficient feature in the code description and the code content. Secondly, a parallel multi-label machine-learning model is developed to process the code description and code content separately. A correlation filer is designed to bridge the two channels to get the relationship between the application background of each code segment and the function modules used in each application. Experiments are conducted in two parts to evaluate the performance of the proposed dataset and the performance of the proposed machine-learning model respectively. The proposed dataset experimental results show that it outperforms dataset with code description only or code content only. In addition, the experiments of the proposed model are carried out on PCD, CDD, and CCD, comparing to the traditional models. The accuracy scores show that the correlation filter can significantly improve the accuracy scores for most of the categories. In conclusion, the proposed dataset structure and the proposed machine-learning model both show promising results comparing to the traditional methodologies.

REFERENCES

[1] Y. Zhou, S. Cui, and Y. Wang, “Development of machine learning based recommendation engine for embedded programmer,” in Proc. IEEE 9th Int. Conf. Electron. Inf. Emergency Commun. (ICEIEC), Beijing, China, Jul. 2019, pp. 634–637.
[2] M. K. Dalal and M. A. Zaveri, “Automatic text classification: A technical review,” Int. J. Comput. Appl., vol. 28, no. 2, pp. 37–40, Aug. 2011.
[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” J. Mach. Learn. Res., vol. 12, pp. 2825–2830, Nov. 2011.
[4] S. Bird, E. Klein, and E. Loper, Natural Language Processing With Python. Sebastopol, CA, USA: O’Reilly, 2009.
[5] S.-B. Kim, K.-S. Han, H.-C. Rim, and S.-H. Myaeng, “Some effective techniques for Naive Bayes text classification,” IEEE Trans. Knowl. Data Eng., vol. 18, no. 11, pp. 1457–1466, Nov. 2006.
[6] G. Ifrim, G. Bakir, and G. Weikum, “Fast logistic regression for text categorization with variable-length n-grams,” in Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), Las Vegas, NV, USA, Aug. 2008, pp. 354–362.
[7] T. Joachims, “Text categorization with support vector machines: Learning with many relevant features,” in Proc. Eur. Conf. Mach. Learn. Chemnitz, Germany: Springer, Apr. 1998, pp. 137–142.
[8] Y. Kim, “Convolutional neural networks for sentence classification,” in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP). Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751.
[9] S. Minee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghl, and J. Gao, “Deep learning-based text classification: A comprehensive review,” ACM Comput. Surv., vol. 54, no. 3, pp. 1–40, Jun. 2021, doi: 10.1145/3439726.
[10] M. H. A. El-Jawad, R. Hodhod, and Y. M. K. Omar, “Sentiment analysis of social media networks using machine learning,” in Proc. 14th Int. Comput. Eng. Conf. (ICENCO), Dec. 2018, pp. 174–176.

[11] X. Luo, “Efficient English text classification using selected machine learning techniques,” Alexandria Eng. J., vol. 60, no. 3, pp. 3401–3409, Jun. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110016821000806

[12] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger, “Tackling the poor assumptions of Naive Bayes text classifiers,” in Proc. 20th Int. Conf. Mach. Learn., vol. 41, Washington, DC, USA, Aug. 2003, pp. 616–623.

[13] Text Classification with Machine Learning & NLP. Accessed: May 11, 2021. [Online]. Available: https://monkeylearn.com/text-classification/

[14] Deep Dive Into Multi-Label Classification! (With Detailed Case Study). Accessed: May 11, 2021. [Online]. Available: https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

[15] M.-L. Zhang and Z.-H. Zhou, “A review on multi-label learning algorithms,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 8, pp. 1819–1837, Aug. 2014.

[16] SVM (Support Vector Machine)—Theory. Accessed: May 11, 2021. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f08126efc72

[17] A. Genkin, D. D. Lewis, and D. Madigan, “Large-scale Bayesian logistic regression for text categorization,” Technometrics, vol. 49, no. 3, pp. 291–304, 2007, doi: 10.1198/004017007000000245.

[18] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in Proc. 28th Int. Conf. Neural Inf. Process. Syst., vol. 1, Montreal, QC, Canada, Dec. 2015, pp. 649–657.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol. (NAACL-HLT), vol. 1, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, MN, USA: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186, doi: 10.18653/v1/n19-1423.

[20] M.-L. Zhang and Z.-H. Zhou, “ML-KNN: A lazy learning approach to multi-label learning,” Pattern Recognit., vol. 40, no. 7, pp. 2038–2048, Jul. 2007.

[21] E. Montañes, R. Senge, J. Barranquero, J. R. Quevedo, J. J. D. Coz, and E. Hüllermeier, “Dependent binary relevance models for multi-label classification,” Pattern Recognit., vol. 47, no. 3, pp. 1494–1508, Mar. 2014.

[22] J. Read, B. Pfahringer, G. Holmes, and E. Frank, “Classifier chains for multi-label classification,” J. Mach. Learn., vol. 85, no. 3, pp. 333–359, 2011.

[23] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, “Learning multi-label scene classification,” Pattern Recognit., vol. 37, no. 9, pp. 1757–1771, Sep. 2004.