EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

Developing a hyperparameter optimization method for classification of code snippets and questions of stack overflow: HyperSCC

Muhammed Maruf Öztürk[1,*]

[1]Department of Computer Engineering, Suleyman Demirel University, West Campus, Isparta, 32040, Turkey

Abstract

Although there exist various machine learning and text mining techniques to identify the programming language of complete code files, multi-label code snippet prediction was not considered by the research community. This work aims at devising a tuner for multi-label programming language prediction of stack overflow posts. To that end, a Hyper Source Code Classifier (HyperSCC) is devised along with rule-based automatic labeling by considering the bottlenecks of multi-label classification. The proposed method is evaluated on seven multi-label predictors to conduct an extensive analysis. The method is further compared with the three competitive alternatives in terms of one-label programming language prediction. HyperSCC outperformed the other methods in terms of the F1 score. Preprocessing results in a high reduction (50%) of training time when ensemble multi-label predictors are employed. In one-label programming language prediction, Gradient Boosting Machine (gbm) yields the highest accuracy (0.99) in predicting R posts that have a lot of distinctive words determining labels. The findings support the hypothesis that multi-label predictors can be strengthened with sophisticated feature selection and labeling approaches.

Received on 21 March 2022; accepted on 26 May 2022; published on 27 May 2022

Keywords: Multi-label classification, hyperparameter optimization, programming language prediction.

doi:10.4108/eai.27-5-2022.174084

1. Introduction

Stack overflow helps software developers to find solutions for a programming problem, thereby including millions of questions and answers. The less experienced programmers are most likely to spend time on stack overflow to enter new questions or review past entries. Those activities do not only increase the speed of development processes but also improve the programming skills of the programmers.

Platforms like quora or stack overflow provide questions and answers along with their tags that ease to find the possible solution. Users are forced to tag their posts in stack overflow. However, inexperienced users may sometimes choose the wrong tags while posting. For instance, the tag "entity-framework" is generally used for both Java and C#. That tag fits very well with C#. Hibernate [1] is a mapping tool that was developed for Java to manipulate objects as in entity-framework. Therefore, a user should employ "Hibernate" to tag object-relational functions in Java rather than "entity-framework". Further, the tag "google-maps" is preferred when the functions related to the mapping are frequently invoked in the applications. However, "google-maps" can not give any hint to figure out the type of programming language. Rather, we expect to see more than one tag to determine exactly what the type of programming language is. Moderators either flag or downvote those posts to cope with misleading information. In that case, the following issues emerge:

1. Wrong tags result in a significant increase in the workload of moderators.
2. Finding a wrongly tagged post becomes difficult even though it presents valuable information.

The majority of the studies finding stack overflow a worthwhile research subject have focused on code analysis [2–4], user behavior [5–7], and predictive

*Corresponding author. Email: muhammedozturk@sdu.edu.tr

models [8–13]. Text mining tools [14, 15] and machine learning (ML) techniques [16–18] are frequently employed in the assessment of stack overflow posts. Further, some researchers [19, 20] investigated to what extent machine learning techniques are beneficial to run a fast and rigorous experiment. Large-scale analysis of the vast data requires a hyperparameter tuning process [21, 22] to obtain reliable results as well.

Integrated Development Environment (IDE) such as Visual Studio, NetBeans, and Xcode allows practitioners to organize and publish their codes. However, those tools can not predict the language of a given file. Rather, they recognize the source code by checking its file extension. This creates a burden for developers editing file extensions manually. To alleviate that burden, software language prediction methods have been developed in various studies [23–25]. However, previous works mostly use data sets including a large number of code lines. ML methods developed for those data sets result in high prediction accuracy since the number of features extracted from the source codes is very high. On the other hand, those methods can not produce promising results when the experimental data sets include relatively small number of code lines.

As stack overflow posts have small code snippets in question blocks, sophisticated code snippet prediction tools are needed. There exist some works [26–28] utilizing tag and question information to predict programming language. Programming Languages Identification (PLI) [29] is the unique commercial tool developed by Algorithmia for predicting programming language of code snippets. PLI claims a high success in programming language prediction (PLP) (>%98 accuracy) but that record was mostly obtained via GitHub codes which are larger than code snippets available in stack overflow.

In this study, multi-label classification of stack overflow questions is conducted. To that end, a novel multi-label label generation method is devised along with a hyperparameter optimization method namely HyperSCC. The method chooses an optimizer by comparing the prediction results obtained through cross-validation on 10% of all training instances. Hence, the most suitable optimizer is met with multi-label predictors that help result in a time-saving experiment.

1.1. Motivation

Multi-language coding is common in software development [30]. In this context, stack overflow questions may have multiple language tags. On the other hand, there is no research on the multi-label classification of code snippets. Preceding works focused on predicting one language tag of the stack overflow questions [27, 31]. In addition to that research gap, there are few researches [20, 32] that analyze the impact of hyperparameter tuning of ML methods handling with stack overflow posts. Hence, disregarding the combination of tuning methods with ML is the main drawback of the preceding works.

The development of hyperparameter tuning techniques has given rise to more precise predictive models [33–38]. However, a tuning process should be organized to conform with experimental data sets [39]. Further, sometimes hyperparameter optimization is suspended or resumed depending on the effectiveness of the tuning process [40]. In this respect, we need new perspectives to improve source code classification techniques.

To clarify the motivation of the paper, Table 1 is designed by summarizing the studies that are similar to our work. Specifically, tag recommendation studies are mostly tuning-free. It is worthwhile to note that this study combines hyperparameter tuning and multi-label prediction.

Apart from preceding works, for multi-label classification, this study presents HyperSCC that alleviates computational burden originated from hyperparameter optimization. Revealing which tuning method is beneficial for programming language prediction helps researchers find strategies to use ML methods in new ways. To the best of our knowledge, this research is the first extensive investigation proposing a tuning approach for the multi-label classification of stack overflow questions.

1.2. Research objectives

This paper defines four research objectives as follows:

Research objective 1 (RO1): Investigate whether automatic rule-based labeling helps increase the success of hyperparameter optimization.
To accomplish this objective, default labels of the posts are modified with a multi-label label data frame. Thereafter, a comparison including seven multi-label predictors is conducted after hyperparameter optimization.
Research objective 2 (RO2): Investigate whether HyperSCC is beneficial for one-label programming language prediction as detected in multi-label prediction.
To accomplish RO2, four state-of-the-art methods including HyperSCC are evaluated with F1 score results.
Research objective 3 (RO3): Examine whether preprocessing helps reduce the training time of multi-label predictors.
For RO3, the training times of multi-label predictors are compared to an increasing number of instances up to 5000.
Research objective 4 (RO4): Examine which script language yields the highest accuracy when using grid search.
For RO4, the accuracy values of eight script languages

Table 1. Comparison of nine similar approaches using stack overflow posts.

Reference	Research question	Hyperparameter tuning	PLP	Multi-label
[18]	Reveal general type of questions on stack overflow	No	No	No
[17]	Explore whether multi-label classifiers are successful in classifying emotions in stack overflow	Yes	No	Yes
[13]	Assess the performance of multi-label predictors in topic recommendation	Yes	No	Yes
[15]	Characterize common architectural design relationships between quality attributes and architectural tactics	No	No	No
[41]	Reveal complaints developer face while using stack overflow	No	No	No
[42]	Develops a tag recommendation framework	No	No	No
[43]	Develops a tag recommendation framework	No	No	No
[31]	Designs a classifier to predict programming language of stack overflow posts	Yes	Yes	No
Our study	Designs a tuned multi-label predictors for PLP	Yes	Yes	Yes

are demonstrated in varying boosting iterations of gbm. Further, eight levels of maximum depth of tree are also analyzed for those iterations.

1.3. Contribution

The major contributions of the study can be elucidated as follows:
1. A hyperparameter tuning method, which utilizes a small part of the training instances to decide the optimizer, is proposed to set hyperparameters of multi-label classification.
2. We develop a rule-based multi-label label generation technique for stack overflow questions.
3. Empirically, to validate the reliability of our method, extensive experiments, which involve single and multiple programming language predictions, are conducted to evaluate and discuss the method.

1.4. Research questions

In this work, four research questions are aimed to be addressed:
RQ1:What type of multi-label classification technique to choose for better success in programming language predictions?
Some multi-label classification techniques produce results depending on label- or size-specific features. Those techniques are discussed and evaluated regarding performance measures in this question.
RQ2: Is HyperSCC superior to the state-of-the-art methods with respect to one-label programming language prediction?
In this question, the advantages and disadvantages of HyperSCC versus the state-of-the-art methods are assessed. 24 programming languages are involved in this sub-experiment.

RQ3: To what extent can preprocessing increase prediction time?
multi-label prediction of programming languages is an effort-intensive and time-consuming process. This question aims to check whether preprocessing reduces training times of the predictors. The preprocessing entails removal of the instances featuring infrequent labels (<5), instances without labels, and constant attributes.
RQ4: Which script language is the most feasible for one-label programming language prediction?
Script languages include similar words. For instance, "array" is one of the most detected words in Perl, PHP, and Lua. It is of great importance to conduct a rigorous analysis of such languages to enhance the comprehensiveness of the study. For this reason, script languages are evaluated both for one-label and multi-label prediction.

The organization of the rest of the article is as follows: Section 2 presents background and notions. The method is described in Section 3. Experimental configurations are presented in Section 4. The findings are given in Section 5 to discuss them in several aspects. Last, the paper is concluded in Section 6.

2. Background

This section describes underlying concept and notions of the study. To this end, four subsections are devised to present a general view.

2.1. Source code classification

The terms 'classification' and 'identification' are sometimes used interchangeably in this field. Source code classification is a challenging issue due to the large number of features extracted from the text corpus.

Let $X = R_m$ denote m-dimensional input space where $y = 0, 1$ is the binary label. Here, the objective is to predict y by utilizing a function $f \rightarrow X$ that maps the input space according to its mathematical assumptions. If the number of instances is very small compared with the m, the prediction may not be completed. On the other hand, m should not be very large to reach prediction results in a reasonable time. To address this problem, feature selection is conducted on R_m. Specifically, X is divided into parts $x_1, ..., x_t$ that the total dimension of these part is equal to m. An $f_r \leftarrow X$ function takes X to delete some dimensions. After that, new dimension can be represented with $m^{'}$ that should meet $m > m^{'}$.

2.2. Multi-label classification

Given an input space $X = R_m$, where $y = y_1, y_2, ..., y_n$ is the label set and n is the number of labels. In programming language classification, n also represents the number of programming languages. Unlike one-label classification, it is necessary to produce multiple outputs for each instance. Hence, performance measures can be extended with hamming-loss, one-error, and subset-accuracy. Feature reduction may also be a feasible solution for multi-label classification. More importantly, infrequent labels should be removed before the training process. Concretely, n is replaced with $n^{''}$ that is obtained with a reduction on n. The threshold of that process depends on the objectives of the established model. In some cases, manual label generation leads to the production of unlabeled instances. To remedy that problem, unlabeled instances are removed. If a feature has a single value for all the instances, it is also removed from X.

2.3. Hyperparameter optimization

Let L be a machine learning algorithm where the parameter and hyperparameter set can be represented with L_p and L_{ph}, respectively. For a tuning function f_t, the main purpose is to configure L_{ph}. On the other hand, L_p is out of scope in that process since the parameters of a machine learning algorithm change during the training. For instance, the weights of a neural network are not tuned since they are determined as constructing the neural network.

If L_{ph} includes three hyperparameters a, b, c in which the length differs depending on the type of hyperparameters. During the tuning process, f_t search a, b, c to find optimal hyperparameter set a_i, b_j, c_k.

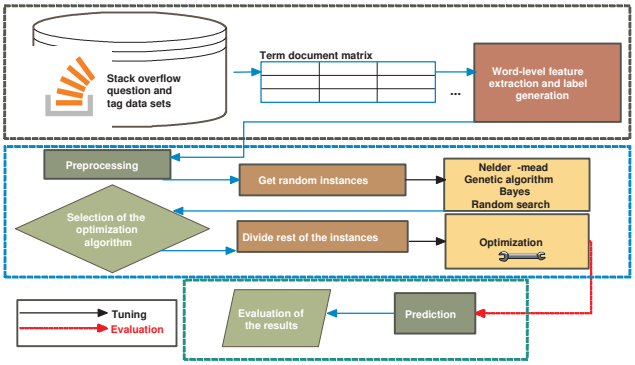


Figure 1. The overview of our approach.

2.4. Problem formulation

Let $L_1, L_2, ..., L_t$ be a set of learning algorithms in which tuning methods $T_1, T_2, ..., T_z$ can be applied to those algorithms. Each tuning algorithm results in a tuning time $\triangle$ along with a performance record P. The main objective is to reveal optimal tuning time $\triangle^*$ that is detected by comparing cost functions produced from $\sum_{i=1}^{z} \triangle_i * P_i$. First and foremost, D_x, which is a small part of training set, is taken from X to analyze $\triangle * P$. Optimal cost function is then represented with $\triangle_o * P_o$ that is calculated for each L. In that comparison, the number of instances should be five or higher times L_{ph}. $\prod_{i=1}^{z} \frac{P_i}{\triangle_i}$ is the general effect of the chosen tuning methods. In this context, the aim of tuning process is to maximize the general effect $max(\prod_{i=1}^{z} \frac{P_i}{\triangle_i})$.

3. HyperSCC

Step 1: Term document matrix generation. HyperSCC starts by analyzing raw data set to extract term document matrix [44] as shown in Figure 1. Each unique word is recognized as a feature in the matrix. Questions and their titles are converted to documents. Texts are interpreted as lower style for each post. After that, punctuation is removed. Numbers and white spaces are eliminated to finalize the document. If those processes are completed successfully, the term document matrix is produced. Word and character counts are further calculated and added to that matrix for each post since they are not available in raw posts.

Step 2: Labeling. Existing works [29, 31] only consider source code language prediction as a binary classification problem. Contrary to these studies, we aim to complete multi-label language prediction for each instance. In labeling, each instance is processed to yield multi-labels representing 24 different programming languages: javascript, sql, java, C#, python, c++, c, php, ruby, swift, objective-c, vb.net, perl, bash, css, scala, html, and lua, haskell, markdown,

R, matlab, GO, and kotlin. In the tag data set, each post may be associated with multiple programming languages. To establish rule-based labeling, at least two distinctive words are searched for each programming language. These words were also extracted in [31]. Further, if a post is tagged with one or more than one of the programming languages, it is labeled as tagged. A labeling algorithm, displayed in Algorithm 1, presents further details about the automatic labeling process. First, the column and row numbers of TDM are calculated to establish a nested loop structure (lines 4-5). The distinctive words determined for programming languages are denoted with $Keywords$ that are set to zero at the beginning of Algorithm 1 (line 7). $y1 : y24$ are the lists created for labeling values of 24 programming languages. $countKeywords$ is the number of distinctive keywords calculated for each programming language, thereby searching TDM (lines 9-11). $ListKeywords$ is generated by converting distinctive keywords to a list (line 13). The tag data set is checked to detect whether the analyzed post is tagged with a specific programming language (line 14). Here, second column of $TagD$ includes tag information ($TagD[, 2]$). If distinctive keywords such as "C#", ".net", "sql", and "php" are identified one or two times in the related instance, it is then labeled as 1. Hence, $threshold$ is either 1 or two, and these values are set depending on the programming language. Lastly, a data frame is generated by using labeling lists $y1 : y24$ to return labeling feature vectors LM.

Algorithm 1 Labeling algorithm

```
1:  Input:TDM(Term document matrix), Tag data set(TagD), threshold
2:  Output:Labeling matrix(LM)
3:  col ← length(TDM[1, ])
4:  row ← length(TDM[, 1])
5:  i, j ← 1
6:  initialize(Keywords, y1 : y24)
7:  while i<row do
8:      while j<col do
9:          countKeywords ← match(TDM[i, j])
10:         j ← j + 1
11:     end while
12:     ListKeywords ← makeList(countKeywords)
13:     if ListKeywords > threshold||tagCalculate(TagD[i, 2] == 1) then
14:         y ← 1
15:     else
16:         y ← 0
17:     end if
18:     i ← i + 1
19:     j ← 1
20: end while
21: LM ← data.frame(y1 : y24)
```

Step 3: Preprocessing. Since each word is regarded as a feature in the posts, feature selection is a must to complete training in a reasonable period. To that end, Pearson correlation analysis is chosen to remove pair-wise correlations. In each step, the correlations are re-evaluated with a specific cutoff (0.7). Character and word counts are not involved in the correlation analysis. The formula of Pearson correlation analysis is given in Equation 1 where a and b denote the variables. $sc(a, b)$ is the sample covariance of them and the sample variances are $sv(a)$ and $sv(b)$. At the end of feature selection, highly correlated features are removed from the term document matrix. Last, the data frame converted from the term document matrix is exposed to a three-phase process. 1) Infrequent labels, which are less than two, are disregarded. 2) The instances having no labels are removed. 3) If a feature has constant value for all the instances, they are also removed from the data frame.

$$R_{ab} = \frac{sc(a, b)}{\sqrt{sv(a).sv(b)}} \quad (1)$$

Step 4: Selection of optimization method. Firstly, 10% of the instances allocated for training are randomly taken. 80% and 20% of the selected instances are used for training and validation, respectively. Thereafter, the results of four optimization methods including Nelder-mead, Genetic algorithm, Bayes, and Random search are evaluated for that validation. They are involved in the experiment due to the following reasons: 1) Nelder-mead requires fewer optimization iterations [45] than the equivalent competitive methods. 2) An intensive data augmentation process is not conducted in Bayesian hyperparameter optimization [46]. 3) Genetic algorithm is a robust hyperparameter optimization technique to reach a well-tuned algorithm to obtain accurate and high results [47]. 4) If the number of hyperparameters is not large, Random search could achieve promising results in a reasonable time [48]. The best method is selected by comparing the prediction accuracies of the optimization methods.

Step 5: Optimization and production of results. In this step, training and testing parts are renewed on the instances (80%-training, and 20%-testing). The training instances are divided into 10 folds. 9 folds are employed for training and one fold is used for the validation. That process is repeated ten times. For each iteration, the validation set is changed. Optimal configuration, which is found by the optimization method decided in the previous step, of the multi-label predictor is saved. 10-fold cross-validation is repeated for the training instances by setting optimal configuration of the multi-label predictors including ensemble of binary relevance (EBR) [49], random k-labelsets (RAKEL) [50], controlled label correlation exploitation (CTRL) [51], ensemble of classifier chain (ECC) [52], ensemble of single label (ESL) [53], label specific features (LIFT) [54], and meta-br (MBR) [55]. Last, the testing set is utilized to yield general prediction results.

4. Experimental settings

4.1. Data sets

We retrieved experimental data sets including three types of data from two sources: questions and tags data

sets that are publicly available [1]. The question data set includes 1264216 posts. Since a post of stack overflow may be tagged with multiple programming languages, the tag data set has 3750994 instances that are relatively higher than that of the question data set. Due to the fact that a concurrent analysis of 20000 instances leads to a huge computational burden (162 GB) for RAM, 500 instances are processed in each iteration of the whole experiment. By averaging the results of those parts, an ultimate output is yielded. The increase in feature number is very fast for first 500 the instances as shown in Figure 3. This is because the number of new words decreases as the number of processed posts increases.

The details of the experimental data sets are given in Table 2. The term document matrix is generated by combining "Title" and "Body" of the Questions data set. The column namely "Sparsity" shows whether the related feature has "NA" values. Note that some questions may remain unclosed so that "ClosedDate" is the sole feature having sparsity. Different from the questions, the answers of stack overflow have no tag as shown in Figure 2. Generally, bodies of questions play an important role in understanding the issue. Questions having an enriched description therefore get a fast and clear response.

4.2. Prediction configurations

The proposed method is coded with R [56] that provides fast computing for machine learning experiments. To run Nelder-mead, nloptr library [57] is utilized. A function namely *neldermead* is run by giving the initial point along with the upper and lower bounds of the target hyperparameter. GA function, which is available in the *GA* library to run the genetic algorithm, is executed with the following configurations: crossover-p: 0.8, mutation-p: 0.1 (p refers probability), and maximum number of iteration: 100. GA is a function of R package GA library [58] that consists of several functions for performing optimizing using genetic algorithms. GA library also enables us to modify genetic operators and run them sequentially or in parallel depending on the experimental design. GA function maximizes a given fitness function using basic principles of genetic algorithm. Parameters that are utilized to run Bayes are as follows: the number of iterations: 50, type of acquisition function: gaussian process upper confidence bound, kappa: 2.576, epsilon of expected improvement, and probability of improvement:0. To perform Random Search, each target hyperparameter is yielded randomly for 100 iterations. Thereafter, the parameter yielding the highest accuracy is determined as the optimal value.

Search spaces of the hyperparameters tuned for multi-label predictors and their definitions are presented in Table 3. It is worthwhile to note that the number of hyperparameters changes depending on the type of predictors. The machine we employ to run the experiment has the following technical properties: 32 CPU(s), Intel(R) Xeon(R) CPU E5-2690, 222 GB Ram, CentOS 7 operating system, and NVIDIA Tesla S870 graphic card. A parallelization is further established on that machine to shorten the completion time of the experiment.

To make a quantitative comparison with other state-of-the-art one-label programming language prediction methods, Xgboost and Random forest algorithms are employed. The mean results of those are compared with those of PLI, SCC [27], SCC++ [31], and DeepSCC [59]. A public key was requested by us to run R script to yield F1 score results of PLI. The R script devised to execute PLI is given in [2].

One-label and multi-label predictions have the same configurations for applying preprocessing and dividing the data sets to obtain performance measures. Three hyperparameters of Xgboost, which inherit the advantages of parallelization in creating decision trees, are subject to optimization as follows: max.depth (the depth of the tree): 3-7, eta (control parameter determining the rate of model learning): 0.001-0.008, nrounds (number of iterations): 19-80. Random forest algorithm is exposed to tuning process to set mtry (number of random variable for each split of the training): 1-5. gbm is employed to compare one-label prediction performances of script languages. Four hyperparameters of gbm are tuned with Grid search algorithm. The hyperparameters and the search space are as follows: interaction.depth:1-8, number of trees: 50-100, learning rate:0.1-0.8, minimum number of observations: 5-20. *caret* library of R is utilized to run gbm along with Grid search.

The source codes of HyperSCC were uploaded to Github, and the URL is https://github.com/muhammedozturk/HyperSCC/. To run HyperSCC, detailed explanations are given in that address. Further, the link of the processed file that was created after feature extraction is available.

4.3. Performance measures

Five performance measures are employed in the experiment: F1 score, accuracy, Hamming-loss, Subset-accuracy, and One-error. However, Hamming-loss [60–62] is the most common evaluation metric of multi-label classification. The formulas of the metrics are given in Equation 2-6, respectively. True positive (TP) is an

[1] https://www.kaggle.com/stackoverflow/stacksample

[2] https://github.com/muhammedozturk/HyperSCC/blob/main/PLI.R

Table 2. Details of the experimental data sets.

Data set	Name	Description	Type	Sparsity
Questions	Id	a primary and incremental key.	integer	no
	OwnerUserId	the number of user who posted the related question.	integer	no
	CreationDate	the date showing when the question is created.	date time	no
	ClosedDate	the date showing when the question is closed.	date time	yes
	Score	a value calculated by (upvotes-downvotes).	integer	no
	Title	a descriptive text of questions.	text	no
	Body	a detailed explanation of questions.	text	no
Tags	Id	a number determines which question is tagged.	integer	no
	Tag	a word labeling questions.	text	no

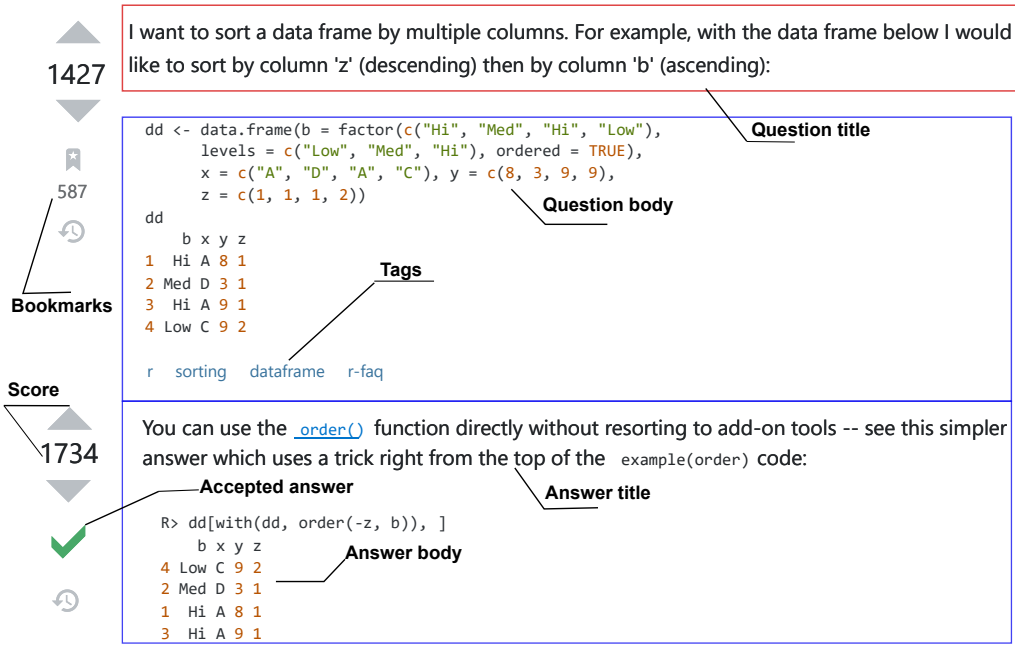


Figure 2. A combination of stack overflow question and answer.

Table 3. Details of the hyperparameters of multi-label predictors. Support Vector Machine (SVM) is set to baseline for the all algorithms.

Method	Hyperparameter	Description	Tuning range
EBR	m	the number of binary relevance models	10-50
	subsample	percentage of training instances	0.1-0.85
	att.space	percentage of attributes	0.1-1
RAKEL	k	the number of labels	3-7
	m	the number of lael powerset models	5-40
CTRL	m	the number of binary relevance models	5-20
	validation.size	the size of validation set	0.1-0.5
	validation.threshold	threshold parameter for determining instance label	0.3-0.7
ECC	m	the number of classififer chain models	10-50
	subsample	percentage of training instances	0.1-1
	attr.space	percentage of attributes	01.-1
ESL	m	the number of members	10-50
	w	the weight of choosing labels	1-5
LIFT	ratio	shows the number of retained clusters	0-1
MBR	folds	the number of folds of internal prediction	1-5
	phi	correlation coefficient	0-1

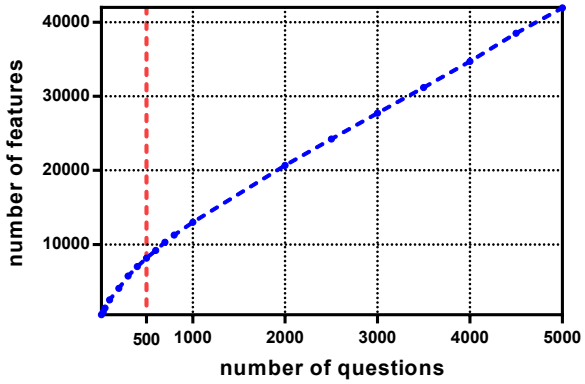


Figure 3. Visualization of the increase in detected features.

indicator that shows the number of correctly predicted positive classes. True negative (TN) is the number of instances that their classes are negative and are correctly predicted. False positive (FP) is the number of wrongly predicted instances that are positive. False negative (FN) is the number of instances that are negative but predicted as positive. F1 score utilizes the elements of the confusion matrix as accuracy does. Hamming-loss shows the ratio of misclassified instances. The extended version of accuracy is called Subset-accuracy that is a harsh metric calculating the most common label. For n observations, here $n.l$ denotes the matrix of label set. R denotes real memberships and P represents predicted memberships. I is the indicator function that evaluates $R_i = P_i$ where R_i and P_i denote the indexes of real and predicted membership values which are being processed at the related iteration. The ratio of irrelevant labels, which is considered as confident, is calculated with One-error.

$$F1 = \frac{TP}{TP + 1/2 * (FP + FN)} \tag{2}$$

$$Accuracy = \frac{TN + TP}{TP + FN + FP + FN} \tag{3}$$

$$Hamming - loss = \frac{1}{n * l} \sum_{i=1}^{n} \sum_{j=1}^{l} (R_i \neq P_j) \tag{4}$$

$$Subset - accuracy = \frac{1}{n} \sum_{i=1}^{n} I * (R_i = P_i) \tag{5}$$

$$One - error = \frac{1}{n} \sum_{i=1}^{n} (maxF(x_i) \notin y_i) \tag{6}$$

5. Results

5.1. RQ1: What type of multi-label classification technique to choose for better success in programming language predictions?

To compare multi-label predictors with respect to HyperSCC, three performance measures presented in Table 4 are produced for each fold of the validations. Note that one-error decreases as the number of validation increases. On the other hand, the results have similar hamming-loss values regardless of the type of predictors. The accuracy results of the multi-label predictors are presented in box-plots in Figure 4. We observe that the predictors show similarities in accuracies as detected in Table 4. Following findings are confirmed with RQ1: 1) The number of validation sets is crucial to stabilize prediction errors. 2) A hyperparameter optimization approach developed for multi-label classifications creates a similar effect on both prediction accuracy and errors.

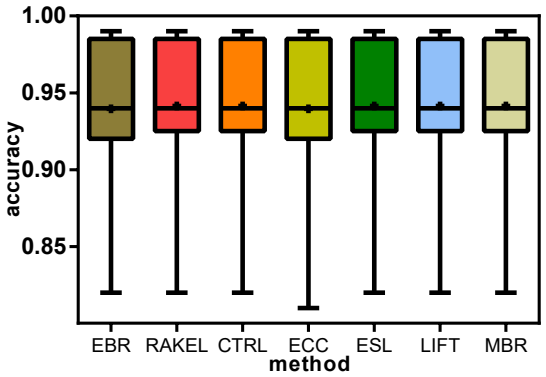


Figure 4. Box-plots of accuracy for multi-label predictors.

5.2. RQ2: Is HyperSCC superior to the state-of-the-art methods with respect to one-label programming language prediction?

To answer this research question, HyperSCC is utilized to obtain combined mean results of XGboost and Random forest. Table 5 reports F1 scores of the state-of-the-art programming language prediction methods along with HyperSCC. It is worthwhile to note that HyperSCC outperformed the others for 15 programming languages. The ineffectiveness of HyperSCC for the other six programming languages may have originated from the labeling rules. GO programming language comprises some common tags such as "api", "sql-server", and "xml" that also belong to other types of programming languages. That case may have led to a dramatic decline in the success of GO prediction. In spite of a lot of syntax and structural properties, F1 score of Java is 0.1 greater than that of C#. The labeling rule of Java has more distinctive words than C# have. For example, the words "java" and "jar" are abundant in stack overflow questions. On the other hand, "instance" is one of the three words utilized in constructing the labeling rule of C#. In that case, the training becomes more imbalanced to learn C# instances. SCC and SC++ are superior to HyperSCC for those languages. More precisely, the

Table 4. The results of HyperSCC of multi-label predictors.

Method	Measure	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
CTRL [51]	hamming-loss	0,0583	0,0571	0,0576	0,0576	0,0571	0,0583	0,0588	0,0592	0,0569	0,0576
	one-error	0,0760	0,0678	0,0657	0,0637	0,0595	0,0575	0,0554	0,0638	0,0679	0,0617
	subset-accuracy	0,3758	0,3717	0,3655	0,3676	0,3676	0,3717	0,3676	0,3683	0,3642	0,3704
ECC [52]	hamming-loss	0,0624	0,0584	0,0587	0,0591	0,0599	0,0595	0,0606	0,0609	0,0591	0,0619
	one-error	0,0657	0,0637	0,0575	0,0616	0,0554	0,0554	0,0534	0,0597	0,0597	0,0556
	subset-accuracy	0,3573	0,3655	0,3593	0,3655	0,3552	0,3676	0,3655	0,3621	0,3704	0,3560
ESL [53]	hamming-loss	0,0584	0,0571	0,0574	0,0576	0,0567	0,0578	0,0590	0,0596	0,0574	0,0579
	one-error	0,0657	0,0637	0,0595	0,0616	0,0554	0,0554	0,0534	0,0597	0,0597	0,0556
	subset-accuracy	0,3696	0,3717	0,3696	0,3676	0,3676	0,3696	0,3655	0,3704	0,3683	0,3683
LIFT [54]	hamming-loss	0,0584	0,0573	0,0574	0,0574	0,0573	0,0586	0,0589	0,0594	0,0571	0,0580
	one-error	0,0678	0,0637	0,0575	0,0616	0,0554	0,0554	0,0534	0,0597	0,0597	0,0576
	subset-accuracy	0,3696	0,3676	0,3696	0,3676	0,3676	0,3676	0,3676	0,3663	0,3663	0,3663
MBR [55]	hamming-loss	0,0446	0,0457	0,0455	0,0454	0,0449	0,0447	0,0452	0,0447	0,0442	0,0444
	one-error	0,0536	0,0639	0,0557	0,0619	0,0577	0,0537	0,0579	0,0517	0,0496	0,0455
	subset-accuracy	0,4763	0,4742	0,4701	0,4742	0,4784	0,4793	0,4752	0,4752	0,4752	0,4752
RAKEL [50]	hamming-loss	0,0578	0,0573	0,0573	0,0578	0,0567	0,0580	0,0590	0,0588	0,0571	0,0578
	one-error	0,0657	0,0637	0,0575	0,0637	0,0554	0,0534	0,0534	0,0597	0,0617	0,0576
	subset-accuracy	0,3717	0,3696	0,3717	0,3676	0,3655	0,3696	0,3676	0,3683	0,3683	0,3683
EBR [49]	hamming-loss	0,0598	0,0594	0,0573	0,0593	0,0579	0,0598	0,0622	0,0614	0,0583	0,0597
	one-error	0,0657	0,0637	0,0575	0,0616	0,0554	0,0554	0,0534	0,0597	0,0597	0,0556
	subset-accuracy	0,3696	0,3655	0,3634	0,3573	0,3593	0,3676	0,3593	0,3580	0,3663	0,3601

Table 5. The comparison of F1 score for four methods. The maximum values for each line are represented with bold text.

language	PLI [29]	SCC [27]	SCC++ [31]	HyperSCC	DeepSCC [59]
Javascript	0.48	0.78	0.74	**0.87**	0.82
SQL	0.5	0.65	0.79	**0.86**	0.81
Java	0.46	0.7	0.76	**0.88**	0.83
C#	0.51	**0.79**	0.78	0.78	0.80
Python	0.69	**0.88**	0.79	0.84	0.81
C++	0.65	0.51	0.53	**0.85**	0.82
C	0.56	0.76	0.81	**0.87**	0.82
PHP	0.62	0.74	**0.88**	0.86	0.81
Ruby	0.43	0.7	0.72	**0.86**	0.79
Swift	0.54	0.84	**0.89**	0.88	0.88
Objective-C	0.77	0.57	0.88	**0.88**	0.84
Vb.Net	0.6	0.83	0.77	**0.89**	0.83
Perl	0.69	0.74	0.41	**0.89**	0.83
Bash	0.67	0.76	**0.85**	0.69	0.83
CSS	0.3	0.86	0.77	**0.88**	0.82
Scala	0.72	0.76	0.81	**0.81**	**0.81**
HTML	0.35	0.54	0.55	**0.88**	0.81
Lua	0.5	0.84	0.7	**0.87**	0.82
Haskell	0.67	0.89	0.78	0.88	**0.89**
Markdown	0.28	0.76	**0.91**	0.87	0.81
R	0.72	0.77	0.78	**0.88**	0.88
Matlab	0.61	0.78	0.79	**0.86**	0.84
GO	0.49	0.67	0.72	0.74	**0.80**
Kotlin	0.60	0.75	0.81	**0.87**	0.83

highest F1 score values of them are not dependent on the types of programming languages. HyperSCC performs well on the instances labeled as Vb.net and Perl. This is because they have more distinctive words than the other programming languages. For instance, one of the top features of Perl is *cgi* which is hardly detected in the posts related to the other programming languages. DeepSCC yielded the highest F1 score for Scala and Haskell. It could not outperform HyperSCC in the rest of the other types of programming languages. Note that DeepSCC had an average of 0.87 F1 score. But it was tested on 179.556 instances [59] which are not greater than that of HyperSCC (1264216). Therefore, employing large data sets result in more realistic performance measures. We performed Wilcoxon Signed-rank Test to check whether there is a noteworthy difference between paired methods. One of

Table 6. Results of Wilcoxon signed rank test of the comparison algorithms with respect to accuracy (H_0:p>0.05, H_1:p<0.05). H_0 is denoted with bold-text.

	PLI	SCC	SCC++	HyperSCC	DeepSCC
PLI	-	0.0002	0.0004	0.0001	0.0003
SCC	0.0002	-	**0.5446**	0.0002	0.00032
SCC++	0.0004	**0.5446**	-	0.0027	0.0037
HyperSCC	0.0001	0.0002	0.0027	-	0.00035
DeepSCC	0.0003	0.00032	0.0037	0.00035	-

the sophisticated types of the t-test is Wilcoxon Signed-rank Test that is generally applied to validate statistical differences between two methods. We do not assume Gaussian distribution among the evaluated methods. H_0 rejects the assumption that the methods yield statistically different results ($p > 0.05$). On the other hand, H_1 is accepted if there is a significant difference between the paired methods ($p < 0.05$). In Table 6, there is a conspicuous difference between HyperSCC and the other methods. However, the statistical test shows that the results of SCC and SCC++ presented in Figure 4 are similar. This is because SCC++ is an improved similar version of SCC.

5.3. RQ3: To what extent can preprocessing increase prediction time?

Preprocessing is generally required in such an experiment to prepare data in a way that guarantees yielding performance measures for each instance. Therefore, exposing data to a preprocessing operation is sometimes crucial. To answer RQ3, a random data set including 5000 instances is retrieved from the data corpus, randomly. A training process is then repeated for the various number of training sizes as follows: 50, 100, 200, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000. At each iteration, the training times of the

multi-label predictors are recorded. Training times can be compared by analyzing the lines presented in Figures 5a and 5b. Notice that there is not any remarkable reduction in training times after preprocessing except for EBR that is the ensemble version of Binary Relevance (BR). In essence, preprocessing should be applied to the algorithms having high training execution time as stated in [63]. LIFT associates the training instances with one of the class labels [54] so that LIFT costs much more time to complete training. On the other hand, MBR needs relatively low training time thanks to its fast decision-making mechanism [55].

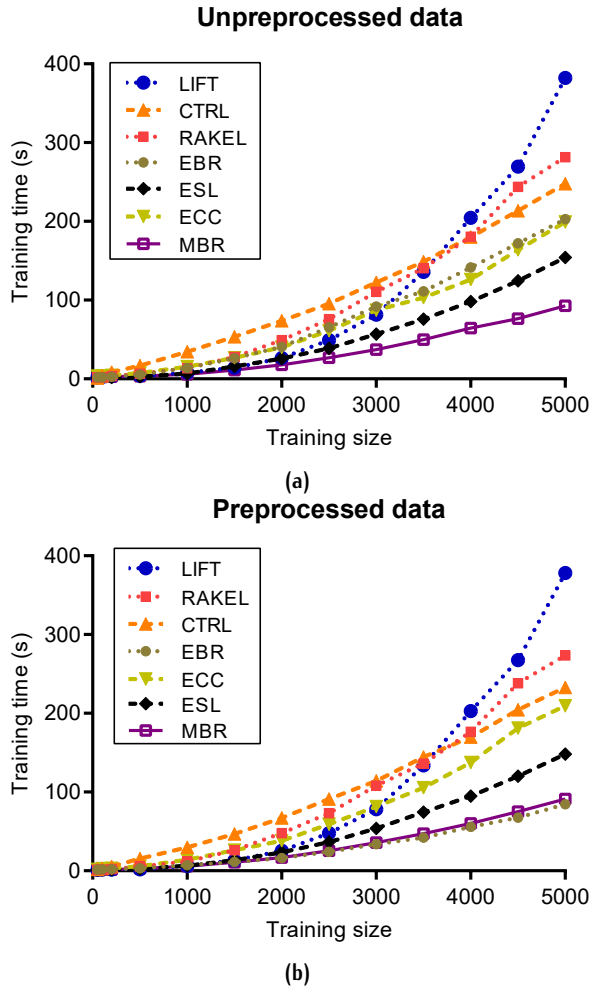


Figure 5. a) Training time curves for unpreprocessed data, b) Training time curves for preprocessed data. Processing is composed of a multi-stage process given in Section 3-Step 3.

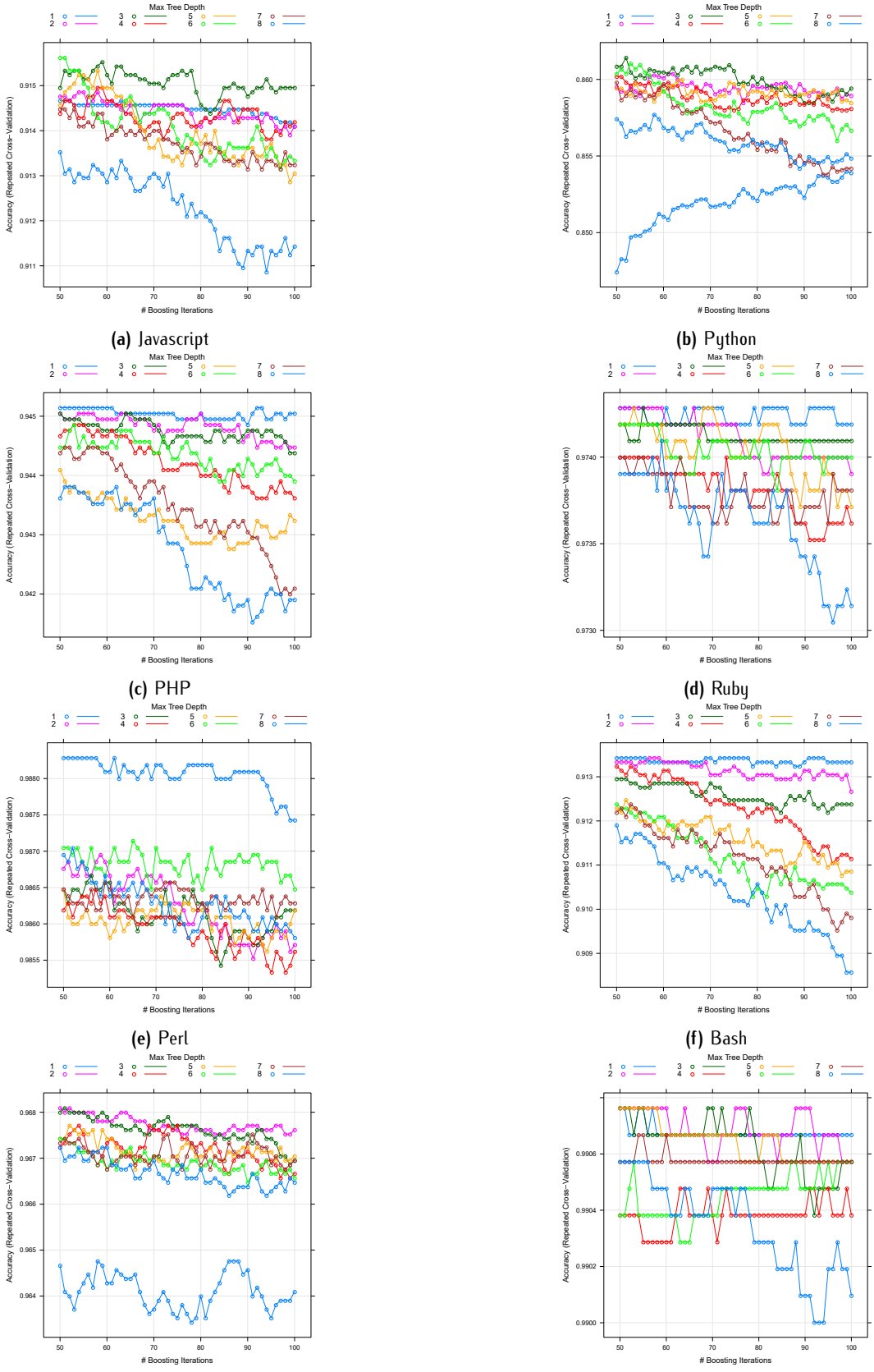
5.4. RQ4: Which script language is the most feasible for predicting one-label programming language?

One-class programming language prediction is performed with gbm to compare the accuracies obtained for eight programming languages including Javascript, Python, PHP, Ruby, Perl, Bash, Lua, and R. Grid search is one of the most popular hyperparameter optimization techniques [64–66] so that we prefer Grid search to perform a straight evaluation of script languages by excluding HyperSCC from this sub-experiment. The details of changing accuracy values are given in Figure 6. The highest accuracy is of the optimal maximum depth of the tree. In this respect, the optimal depth of the tree (1) of Perl is quite distinct from all of the others. R outperformed the other script languages that it yielded 0.99 of accuracy. The accuracy of Python is the lowest (0.89) among those which produced accuracy values higher than 0.9. Fluctuations in the accuracy of R are vastly clearer than those of the others. Moreover, optimizing the maximum depth of the tree is easier for Perl which has a large margin between the optimal (1) and ordinary hyperparameters (2-3-4-5-6-7-8).

5.5. Discussion and implications

In this section, we discuss to what extent this study differs from existing works by delving into the results presented in the previous sections.

Programming language prediction techniques suffer from various data-centric and experimental drawbacks. The majority of studies focus either on creating a feature set from the raw text [27] or the robustness of machine learning techniques [59, 67]. However, relying on programming language labels of data sets without making an in-depth tag analysis may result in unreliable predictions. The reason is that tagging is a compulsory operation, especially for posting questions in stack overflow. The automatic labeling process presented in Section 3 aims to enhance the reliability of labels of training data. The results presented in Table 2 support the hypothesis that a suitable process made on labeling features has a positive impact on prediction success. Stack overflow data sets had been exposed to the machine learning process by using the same hyperparameter set employed in the previous studies[68] or utilizing one hyperparameter tuning technique [69]. Instead, we have conducted a mini-batch training to decide the tuning approach to be used in the rest of the experiment. In this way, we were able to produce high accuracy regardless of the type of predictors in multi-label programming language prediction. Apart from preceding studies, the results of this study assert that performing one-label programming language prediction on Perl and R yields higher accuracy than that of other script languages. This validates that the programming languages including less distinctive keywords do not respond as well to labeling and preprocessing processes.



(a) Javascript **(b)** Python

(c) PHP **(d)** Ruby

(e) Perl **(f)** Bash

(g) Lua **(h)** R

Figure 6. Accuracy curves of gbm for various boosting iterations.

6. Conclusion and Future Remarks

In this work, we proposed a new hyperparameter optimization technique namely HyperSCC for multi-label code prediction of stack overflow, where wrong labeling is addressed by establishing an automatic multi-labeling process. Compared with the state-of-the-art source code classification methods, the experimental steps we present in the paper do not rely on a specific optimization technique. Instead, the best optimization technique is chosen by analyzing the performance of a smaller part of the training data set, thereby executing HyperSCC. According to the obtained findings; 1) The success of multi-label source code prediction is not related to the predictor. 2) For one-label prediction, HyperSCC is superior to the other three methods in the F1 score. 3)The time passed for the training mainly depends on the type of multi-label predictor. MBR shows significant resistance to the increase of training time for a large number of training instances. Further, the effect of data processing in training time is negligible for the majority of the multi-label predictors. 4) The tuning burden of predicting script languages can be alleviated via more robust labeling and tagging approaches.

This paper can be extended with the future agenda encompassing the following items: 1) Raw posts of stack overflow create a remarkable computational burden if each word is recognized as a feature as performed in the experiment. We rather need to develop feature selection techniques, thereby regarding exceptional cases of source code prediction. 2) The labeling method presented in this work could be leveraged by establishing a fuzzy rule-based model [70]. 3) The effectiveness of HyperSCC may be validated by comparing with the methods developed for distributed data optimization [71] and resource allocation issues [72].

Declarations

Funding Not applicable.

Conflict of interest The authors declare that they have no confict of interest.

Availability of data and material The data required to replicate the experiment is presented in Section 5.2.

Code availability The link required to access the replication packages is presented in Appendix A.

Authors' contributions Not applicable.

Ethics approval This article dos not contain any studies with human participants or animals performed by any of the authors.

Consent to participate Informed consent was obtained from all individual participants included in the study.

Consent for publication The authors affirm that human research participants provided informed consent for publication.

References

[1] Huang, Z., Shao, Z., Fan, G., Yu, H., Yang, K. and Zhou, Z. (2022) Hbsniff: A static analysis tool for java hibernate object-relational mapping code smell detection. *Science of Computer Programming* **217**: 102778.

[2] Nasehi, S.M., Sillito, J., Maurer, F. and Burns, C. (2012) What makes a good code example?: A study of programming q&a in stackoverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)* (IEEE): 25–34.

[3] Linares-Vásquez, M., Bavota, G., Di Penta, M., Oliveto, R. and Poshyvanyk, D. (2014) How do api changes trigger stack overflow discussions? a study on the android sdk. In *proceedings of the 22nd International Conference on Program Comprehension*: 83–94.

[4] Abdalkareem, R., Shihab, E. and Rilling, J. (2017) On code reuse from stackoverflow: An exploratory study on android apps. *Information and Software Technology* **88**: 148–158.

[5] Yang, J., Tao, K., Bozzon, A. and Houben, G.J. (2014) Sparrows and owls: Characterisation of expert behaviour in stackoverflow. In *International conference on user modeling, adaptation, and personalization* (Springer): 266–277.

[6] Ahmed, T. and Srivastava, A. (2017) Understanding and evaluating the behavior of technical users. a study of developer interaction at stackoverflow. *Human-centric Computing and Information Sciences* **7**(1): 1–18.

[7] Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R. and Lanza, M. (2014) Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th working conference on mining software repositories*: 102–111.

[8] Baltadzhieva, A. and Chrupała, G. (2015) Predicting the quality of questions on stackoverflow. In *Proceedings of the international conference recent advances in natural language processing*: 32–40.

[9] Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K. and Schneider, K.A. (2018) Classifying stack overflow posts on api issues. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)* (IEEE): 244–254.

[10] Tóth, L., Nagy, B., Janthó, D., Vidács, L. and Gyimóthy, T. (2019) Towards an accurate prediction of the question quality on stack overflow using a deep-learning-based nlp approach. In *ICSOFT*: 631–639.

[11] Neshati, M. (2017) On early detection of high voted Q&A on Stack Overflow. *Information Processing & Management* **53**(4): 780–798. doi:10.1016/j.ipm.2017.02.005, URL https://linkinghub.elsevier.com/retrieve/pii/S0306457316304733.

[12] Amancio, L., Dorneles, C.F. and Dalip, D.H. (2021) Recency and quality-based ranking question in CQAs: A Stack Overflow case study. *Information Processing & Management* **58**(4): 102552. doi:10.1016/j.ipm.2021.102552, URL

https://linkinghub.elsevier.com/retrieve/pii/S030645732100056X.

[13] Izadi, M., Heydarnoori, A. and Gousios, G. (2021) Topic recommendation for software repositories using multi-label classification algorithms. *Empirical Software Engineering* **26**(5): 93. doi:10.1007/s10664-021-09976-2, URL https://link.springer.com/10.1007/s10664-021-09976-2.

[14] Joorabchi, A., English, M. and Mahdi, A.E. (2016) Text mining stackoverflow: An insight into challenges and subject-related difficulties faced by computer science learners. *Journal of Enterprise Information Management* .

[15] Bi, T., Liang, P., Tang, A. and Xia, X. (2021) Mining Architecture Tactics and Quality Attributes knowledge in Stack Overflow. *Journal of Systems and Software* **180**: 111005. doi:10.1016/j.jss.2021.111005, URL https://linkinghub.elsevier.com/retrieve/pii/S0164121221001023.

[16] Alshangiti, M., Sapkota, H., Murukannaiah, P.K., Liu, X. and Yu, Q. (2019) Why is developing machine learning applications challenging? a study on stack overflow posts. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (IEEE): 1–11.

[17] Cabrera-Diego, L.A., Bessis, N. and Korkontzelos, I. (2020) Classifying emotions in Stack Overflow and JIRA using a multi-label approach. *Knowledge-Based Systems* **195**: 105633. doi:10.1016/j.knosys.2020.105633, URL https://linkinghub.elsevier.com/retrieve/pii/S0950705120300939.

[18] Beyer, S., Macho, C., Di Penta, M. and Pinzger, M. (2020) What kind of questions do developers ask on Stack Overflow? A comparison of automated approaches to classify posts into question categories. *Empirical Software Engineering* **25**(3): 2258–2301. doi:10.1007/s10664-019-09758-x, URL http://link.springer.com/10.1007/s10664-019-09758-x.

[19] Bangash, A.A., Sahar, H., Chowdhury, S., Wong, A.W., Hindle, A. and Ali, K. (2019) What do developers know about machine learning: a study of ml discussions on stackoverflow. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (IEEE): 260–264.

[20] Menzies, T., Majumder, S., Balaji, N., Brey, K. and Fu, W. (2018) 500+ times faster than deep learning:(a case study exploring faster methods for text mining stackoverflow). In *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)* (IEEE): 554–563.

[21] Tellez, E.S., Moctezuma, D., Miranda-Jiménez, S. and Graff, M. (2018) An automated text categorization framework based on hyperparameter optimization. *Knowledge-Based Systems* **149**: 110–123.

[22] Probst, P., Wright, M.N. and Boulesteix, A.L. (2019) Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery* **9**(3): e1301.

[23] Van Dam, J.K. and Zaytsev, V. (2016) Software language identification with natural language classifiers. In *2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER)* (IEEE), **1**: 624–628.

[24] Gilda, S. (2017) Source code classification using neural networks. In *2017 14th international joint conference on computer science and software engineering (JCSSE)* (IEEE): 1–6.

[25] Barchi, F., Parisi, E., Urgese, G., Ficarra, E. and Acquaviva, A. (2021) Exploration of convolutional neural network models for source code classification. *Engineering Applications of Artificial Intelligence* **97**: 104075.

[26] Baquero, J.F., Camargo, J.E., Restrepo-Calle, F., Aponte, J.H. and González, F.A. (2017) Predicting the programming language: Extracting knowledge from stack overflow posts. In *Colombian Conference on Computing* (Springer): 199–210.

[27] Alreshedy, K., Dharmaretnam, D., German, D.M., Srinivasan, V. and Gulliver, T.A. (2018) Scc: automatic classification of code snippets. *arXiv preprint arXiv:1809.07945* .

[28] Singh, P., Chopra, R., Sharma, O. and Singla, R. (2020) Stackoverflow tag prediction using tag associations and code analysis. *Journal of Discrete Mathematical Sciences and Cryptography* **23**(1): 35–43.

[29] Programming language identification tool, https://algorithmia.com/algorithms/PetiteProgrammer/ProgrammingLanguageIdentification. Accessed: 2022-03-10.

[30] Mayer, P., Kirsch, M. and Le, M.A. (2017) On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers. *Journal of Software Engineering Research and Development* **5**(1): 1. doi:10.1186/s40411-017-0035-z, URL http://jserd.springeropen.com/articles/10.1186/s40411-017-0035-z.

[31] Alrashedy, K., Dharmaretnam, D., German, D.M., Srinivasan, V. and Gulliver, T.A. (2020) Scc++: Predicting the programming language of questions and snippets of stack overflow. *Journal of Systems and Software* **162**: 110505.

[32] Rosen, C. and Shihab, E. (2016) What are mobile developers asking about? a large scale study using stack overflow. *Empirical Software Engineering* **21**(3): 1192–1223.

[33] Stoddard, J.G., Welsh, J.S. and Hjalmarsson, H. (2017) Em-based hyperparameter optimization for regularized volterra kernel estimation. *IEEE control systems letters* **1**(2): 388–393.

[34] Thiede, L.A. and Parlitz, U. (2019) Gradient based hyperparameter optimization in echo state networks. *Neural Networks* **115**: 23–29.

[35] Mantovani, R.G., Rossi, A.L., Alcobaca, E., Vanschoren, J. and de Carvalho, A.C. (2019) A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves svm classifiers. *Information Sciences* **501**: 193–221.

[36] Minku, L.L. (2019) A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. *Empirical Software Engineering* **24**(5): 3153–3204.

[37] Schratz, P., Muenchow, J., Iturritxa, E., Richter, J. and Brenning, A. (2019) Hyperparameter tuning and performance assessment of statistical and machine-learning

algorithms using spatial data. *Ecological Modelling* **406**: 109–120.

[38] Osman, H., Ghafari, M. and Nierstrasz, O. (2017) Hyperparameter optimization to improve bug prediction accuracy. In *2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)* (IEEE): 33–38.

[39] Agrawal, A., Yang, X., Agrawal, R., Yedida, R., Shen, X. and Menzies, T. (2021) Simpler Hyperparameter Optimization for Software Analytics: Why, How, When. *IEEE Transactions on Software Engineering* : 1–1doi:10.1109/TSE.2021.3073242, URL https://ieeexplore.ieee.org/document/9405415/.

[40] Tran, N., Schneider, J.G., Weber, I. and Qin, A.K. (2020) Hyper-parameter optimization in classification: To-do or not-to-do. *Pattern Recognition* **103**: 107245.

[41] Cummaudo, A., Vasa, R., Barnett, S., Grundy, J. and Abdelrazek, M. (2020) Interpreting cloud computer vision pain-points. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (New York, NY, USA: ACM): 1584–1596. doi:10.1145/3377811.3380404, URL https://dl.acm.org/doi/10.1145/3377811.3380404.

[42] Maity, S.K., Panigrahi, A., Ghosh, S., Banerjee, A., Goyal, P. and Mukherjee, A. (2019) DeepTagRec: A Content-cum-User Based Tag Recommendation Framework for Stack Overflow. 125–131. doi:10.1007/978-3-030-15719-7_16, URL http://link.springer.com/10.1007/978-3-030-15719-7{_}16.

[43] Wang, H., Wang, B., Li, C., Xu, L., He, J. and Yang, M. (2019) SOTagRec. In *Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence - ICMAI 2019* (New York, New York, USA: ACM Press): 146–152. doi:10.1145/3325730.3325751, URL http://dl.acm.org/citation.cfm?doid=3325730.3325751.

[44] Feinerer, I. (2013) Introduction to the tm package text mining in r. *Accessible en ligne: http://cran. r-project. org/web/packages/tm/vignettes/tm. pdf* .

[45] Yoshihiko, O., Shuhei, W. and Masaki, O. (2019) Accelerating the nelder-mead method with predictive parallel evaluation. In *6th ICML Workshop on Automated Machine Learning (AutoML2019)*.

[46] Klein, A., Falkner, S., Bartels, S., Hennig, P. and Hutter, F. (2017) Fast Bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics* **11**(2). doi:10.1214/17-EJS1335SI, URL https://projecteuclid.org/journals/electronic-journal-of-statistics/volume-11/issue-2/Fast-Bayesian-hyperparameter-optimization-on-large-datasets/10.1214/17-EJS1335SI.full.

[47] Daviran, M., Maghsoudi, A., Ghezelbash, R. and Pradhan, B. (2021) A new strategy for spatial predictive mapping of mineral prospectivity: Automated hyperparameter tuning of random forest approach. *Computers & Geosciences* **148**: 104688. doi:10.1016/j.cageo.2021.104688, URL https://linkinghub.elsevier.com/retrieve/pii/S0098300421000030.

[48] Balaprakash, P., Salim, M., Uram, T.D., Vishwanath, V. and Wild, S.M. (2018) DeepHyper: Asynchronous Hyperparameter Search for Deep Neural Networks. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)* (IEEE): 42–51. doi:10.1109/HiPC.2018.00014, URL https://ieeexplore.ieee.org/document/8638041/.

[49] Read, J., Pfahringer, B., Holmes, G. and Frank, E. (2011) Classifier chains for multi-label classification. *Machine Learning* **85**(3): 333–359. doi:10.1007/s10994-011-5256-5, URL http://link.springer.com/10.1007/s10994-011-5256-5.

[50] Tsoumakas, G, K.I.V.I. (2010) Random k-labelsets for multilabel classification. *IEEE transactions on knowledge and data engineering* **23**(7): 1079–1089.

[51] Li, Y.K. and Zhang, M.L. (2014) Enhancing binary relevance for multi-label learning with controlled label correlations exploitation. In *Pacific Rim International Conference on Artificial Intelligence* (Springer): 91–103.

[52] Senge, R., del Coz, J.J. and Hüllermeier, E. (2014) On the Problem of Error Propagation in Classifier Chains for Multi-label Classification. 163–170. doi:10.1007/978-3-319-01595-8_18, URL http://link.springer.com/10.1007/978-3-319-01595-8{_}18.

[53] Wang, R., Kwong, S., Wang, X. and Jia, Y. (2021) Active k-labelsets ensemble for multi-label classification. *Pattern Recognition* **109**: 107583. doi:10.1016/j.patcog.2020.107583, URL https://linkinghub.elsevier.com/retrieve/pii/S0031320320303861.

[54] Zhang, M.L. and Wu, L. (2014) Lift: Multi-label learning with label-specific features. *IEEE transactions on pattern analysis and machine intelligence* **37**(1): 107–120.

[55] Read, J., Puurula, A. and Bifet, A. (2014) Multi-label classification with meta-labels. In *2014 IEEE international conference on data mining* (IEEE): 941–946.

[56] Team, R.C. (2000) R language definition. *Vienna, Austria: R foundation for statistical computing* .

[57] Ypma, J. (2014) Introduction to nloptr: an r interface to nlopt. *R Package* **2**.

[58] Scrucca, L. (2013) Ga: a package for genetic algorithms in r. *Journal of Statistical Software* **53**: 1–37.

[59] Yang, G., Zhou, Y., Yu, C. and Chen, X. (2021) DeepSCC: Source Code Classification Based on Fine-Tuned RoBERTa URL http://arxiv.org/abs/2110.00914. 2110.00914.

[60] Liao, W., Wang, Y., Yin, Y., Zhang, X. and Ma, P. (2020) Improved sequence generation model for multi-label classification via cnn and initialized fully connection. *Neurocomputing* **382**: 188–195.

[61] Nguyen, V.L. and Hullermeier, E. (2020) Reliable multilabel classification: Prediction with partial abstention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, **34**: 5264–5271.

[62] Nazmi, S., Yan, X., Homaifar, A. and Doucette, E. (2020) Evolving multi-label classification rules by exploiting high-order label correlations. *Neurocomputing* **417**: 176–186.

[63] Moyano, J.M., Gibaja, E.L., Cios, K.J. and Ventura, S. (2018) Review of ensembles of multi-label classifiers: models, experimental study and prospects. *Information Fusion* **44**: 33–45.

[64] Bertrand, Q., Klopfenstein, Q., Blondel, M., Vaiter, S., Gramfort, A. and Salmon, J. (2020) Implicit

differentiation of lasso-type models for hyperparameter optimization. In III, H.D. and Singh, A. [eds.] *Proceedings of the 37th International Conference on Machine Learning* (PMLR), *Proceedings of Machine Learning Research* **119**: 810–821. URL https://proceedings.mlr.press/v119/bertrand20a.html.

[65] Stuke, A., Rinke, P. and Todorović, M. (2021) Efficient hyperparameter tuning for kernel ridge regression with bayesian optimization. *Machine Learning: Science and Technology* **2**(3): 035022. doi:10.1088/2632-2153/abee59, URL https://doi.org/10.1088/2632-2153/abee59.

[66] Tu, H. and Nair, V. (2018) Is one hyperparameter optimizer enough? In *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics* (New York, NY, USA: ACM): 19–25. doi:10.1145/3278142.3278145, URL https://dl.acm.org/doi/10.1145/3278142.3278145.

[67] Kiyak, E.O., Cengiz, A.B., Birant, K.U. and Birant, D. (2020) Comparison of Image-Based and Text-Based Source Code Classification Using Deep Learning. *SN Computer Science* **1**(5): 266. doi:10.1007/s42979-020-00281-1, URL https://link.springer.com/10.1007/s42979-020-00281-1.

[68] Xu, B., Hoang, T., Sharma, A., Yang, C., Xia, X. and Lo, D. (2021) Post2Vec: Learning Distributed Representations of Stack Overflow Posts. *IEEE Transactions on Software Engineering* : 1–1doi:10.1109/TSE.2021.3093761, URL https://ieeexplore.ieee.org/document/9469219/.

[69] Cao, K., Chen, C., Baltes, S., Treude, C. and Chen, X. (2021) Automated Query Reformulation for Efficient Search Based on Query Logs From Stack Overflow. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (IEEE): 1273–1285. doi:10.1109/ICSE43902.2021.00116, URL https://ieeexplore.ieee.org/document/9402151/.

[70] Stepin, I., Alonso, J.M., Catala, A. and Pereira-Farina, M. (2020) Generation and evaluation of factual and counterfactual explanations for decision trees and fuzzy rule-based classifiers. In *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (IEEE): 1–8. doi:10.1109/FUZZ48607.2020.9177629, URL https://ieeexplore.ieee.org/document/9177629/.

[71] Ge, Y.F., Orlowska, M., Cao, J., Wang, H. and Zhang, Y. (2022) Mdde: multitasking distributed differential evolution for privacy-preserving database fragmentation. *The VLDB Journal* : 1–19.

[72] Li, J.Y., Du, K.J., Zhan, Z.H., Wang, H. and Zhang, J. (2022) Distributed differential evolution with adaptive resource allocation. *IEEE Transactions on Cybernetics* .