electronics

MDPI

Article

Software Subclassification Based on BERTopic-BERT-BiLSTM Model

Wenjuan Bu, Hui Shu *, Fei Kang, Qian Hu and Yuntian Zhao

School of Cybersecurity, PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China; buwenjuan_521@163.com (W.B.); kfminnie@163.com (F.K.); 13164379786@163.com (Q.H.); yuntianzhao1105@163.com (Y.Z.)
* Correspondence: shuhui123@126.com

Abstract: With the continuous influx of application software onto the application software market, achieving accurate software recommendations for users in the huge software application market is urgent. To address this issue, each application software market currently provides its own classification tags. However, several problems still exist, such as the lack of objectivity, hierarchy, and standardization in these classifications, which in turn affects the accuracy of precise software recommendations. Accordingly, a customized BERTopic model is proposed to cluster the software description texts of the application software and the automatic tagging and updating of the application software tags are realized according to the clusters obtained by topic clustering and the extracted subject words. At the same time, a data enhancement method based on the c-TF-IDF algorithm is proposed to solve the problem of imbalance of datasets, and then the classification model based on the BERT-BiLSTM model is trained on the labeled datasets to classify the software in the dimension of the application function, so as to realize the accurate software recommendation for users. Based on the experimental verification of two datasets, 21 categories in the SourceForge dataset and 19 categories in the Chinese App Store dataset are subclassed by the clustering results of the customized BERTopic model, and the tags of 138 subclasses and 262 subclasses are formed, respectively. In addition, a complete tagged software description text dataset is constructed and the software tags are updated automatically. In the first stage of the classification experiment, the weighted average accuracy, recall rate, and F1 value can reach 0.92, 0.91, and 0.92, respectively. In the second stage, the weighted average accuracy, recall rate, and F1 value can all reach 0.96. After data enhancement, the weighted average F1 value of the classification model can be increased by up to two percentage points.

Keywords: application software; BERTopic; BERT; BiLSTM; c-TF-IDF algorithm

check for updates

Citation: Bu, W.; Shu, H.; Kang, F.; Hu, Q.; Zhao, Y. Software Subclassification Based on BERTopic-BERT-BiLSTM Model. *Electronics* **2023**, *12*, 3798. https://doi.org/10.3390/electronics12183798

Academic Editor: Alberto Fernandez Hilario

Received: 13 July 2023
Revised: 4 September 2023
Accepted: 6 September 2023
Published: 8 September 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

As new applications continue to flood onto the application software market, it has developed into a key concern on the application software market as to how to quickly recommend applications according to the user's interests and preferences among a large amount of application software. The application software market, such as SourceForge and Google Play, provides a convenient platform for developers to promote products and users to search and download applications. According to statistics, there are more than 3 million existing applications and 600,000 new applications stationed in the platform every year [1]. In the growing application software market, we try to build fine-grained category tags based on the functional dimensions of application software. The automatic subclassification of large-scale application software may be a feasible scheme to solve the problem of the accurate application of software recommendations. Each existing application market provides its application classification tag, but there are still several potential problems:

First, the existing classification lacks objectivity. Application software is usually classified according to the category or software application description text specified by

application developers, but in many cases, the software application description text is very short and easy to write, and the description of the same application software is different for each application market. Additionally, a significant number of software developers either fail to specify the software application category or provide vague descriptions. Moreover, the software application descriptions provided by developers often differ from the actual functionality of the application software. Correspondingly, there is a large amount of application software on the application software market, and it is not feasible to implement strict manual guidance or revisions to the application software classification label. As a result, it is almost impossible for each software application market to achieve an accurate classification of application software, and the current application software classification methods used in the application market cannot completely guarantee the accuracy of application software category labels.

Second, the existing classification lacks hierarchy. For example, the Google Play market provides 25 categories of flat classification tags, except for the hierarchical classification of game categories, divided into 6 subclasses [2]; the commonly used software butler in China divides applications into 25 categories according to their functions, although 20 of them are classified hierarchically [3], but the accuracy of subclass labels and subclasses of applications is significantly reduced. According to the investigation and analysis, other application software markets have also adopted a similar, basically flat classification and label system, with the explosive growth in application software. There are an average of 15,000 applications in each category [2]. This coarse-grained and flattened classification cannot effectively distinguish applications and makes it time-consuming and laborious for users to search for applications that they are interested in. This indirectly implies that how to automatically hierarchize coarse-grained flat classification tags into fine-grained subclass tags in the application software market is still a problem to be studied.

Third, the existing classification lacks standards. The category labels and the number of categories vary for each application software market due to the absence of a unified standard in the classification system. These category labels are usually determined by platform managers. Furthermore, application markets often restrict the classification of applications to a single category to prevent software developers from engaging in excessive advertising by assigning applications to multiple categories, as highlighted in Reference [4]. Rigid and exclusive tags will cause a large number of multi-category applications to be lost from the corresponding categories. There is no unified standard in the classification method of the application store to determine the categories of applications with multiple categories of functions, and most of them rely on the subjective judgment of platform managers; on the other hand, new types of applications continue to emerge in the application market, and how to realize the automatic label classification of new categories of applications according to the software application description is also a problem to be solved in the application market.

In a word, the current application software classification methods for the application software market are still faced with massive challenges. To effectively guide the construction of the application market classification label and the automatic fine-grained classification of the application software in the application function dimension, and to realize the accurate application software recommendation according to the user's interest preference, a software classification framework based on the BERTopic-BERT-BiLSTM model is proposed, which takes software description text as the research object, and realizes the subdivision of the application software functional dimension by combining a clustering model and a classification model.

The Bidirectional Encoder Representations from Transformers (BERT) [5], is a large-scale pre-trained language model capable of word vector representation of software description text, capturing the semantic and contextual information of words in software description texts in their context through the transformer-based encoder structure. The contextually sensitive, enabling BERT model enhances the comprehension of word meanings and relationships within the software's description text, ultimately leading to an improved understanding. Consequently, it provides a more precise textual representation.

The Bidirectional Long Short-Term Memory (BiLSTM) [6] utilizes both the forward and backward networks to encode contextual information at the current moment. By capturing long-sequence dependencies in the sequence of software-described text vectors generated by the BERT model, BiLSTM enables a deeper understanding of semantic information within the software-described text.

The BERTopic model performs cluster analysis based on word vector text and uses Cluster-based Term Frequency–Inverse Document Frequency (c-TF-IDF) to extract subject words in clustered clusters [7]. It should be emphasized that the clustered software description text may have different clustering topics, while the c-TF-IDF algorithm can extract subject terms based on a specific set of texts. Compared with the traditional TF-IDF algorithm, c-TF-IDF can better reflect the characteristics of the clustering topic and extract keywords related to the topic on the specific clustering results. The main contributions are as follows:

1. We have developed a customized BERTopic topic clustering model specifically designed for software description text;
2. We propose a data augmentation method based on the c-TF-IDF algorithm;
3. We propose a fine-grained software classification model, BERT-BiLSTM, which is validated through two-stage experiments on two different datasets.

2. Research on Related Methods

In this section, we will elucidate the research methodologies employed in this study. We will delve into the software classification method, text classification technology, data enhancement techniques, and the BERTopic topic clustering model.

2.1. Software Classification Method

In the early stage, the machine learning method is mainly used to classify the application software according to the text description information of the application software. Typical machine learning algorithms such as Support Vector Machine (SVM) and Naive Bayes are used to classify the features of the software text description information. For example, Reference [8] designed an effective combination strategy to combine the online configuration files of multiple software repositories, and the SVM algorithm was used as a classifier to classify the combined online configuration files. Finally, 123 categories of application software are classified, and the overall accuracy, recall rate, and F-measure can reach 71.41%, 65.6%, and 68.38%. Reference [9] proposed using the Naive Bayes algorithm to construct a classifier to classify the applications in the Google Play application store. When all game categories are consolidated into a single category, resulting in a total of 22 categories, the classifier attains an accuracy of 87%. However, when the game category is divided into 15 subcategories, the classifier's performance decreases to 72.7%. The main reason is that the application software in each game category often uses similar words for descriptions. The classifier fails to recognize these similar words correctly, which causes the application to be misclassified.

Clustering algorithms in machine learning are also widely used in software classification. For example, MUDABlue [10] is a software classification system that incorporates Latent Semantic Analysis (LSA) for classifying software based on a source code. Additionally, it has the capability to automatically generate software categories while accommodating the possibility of software belonging to multiple categories. Reference [11] analyzed software categories through the probabilistic topic model Latent Dirichlet Allocation (LDA), which treated each software type as a document composed of a set of words, including code identifiers and comments parsed from source code, and classified software based on similarity cluster topics. Compared with Reference [10], the proposed method generates more accurate category tags. Reference [12] proposed Label Software Topic Detection (LSTD), which detected and enriched the software topics by mining a large number of text software configuration files combining the theme model LDA and the sorting mechanism, which realized the classification and label recommendation tasks of the software. Under

the guidance of fine-grained hierarchical ontology, Reference [2] classified the applications in the Google Play application store through the semi-supervised clustering algorithm Non-negative Matrix Factorization (NMF), constructed a hierarchical fine-grained classification framework for applications, and implemented 49 categories of software fine-grained hard classification, with an accuracy of 83.2% at 100% coverage.

With the rapid development of deep learning and natural language processing technology, deep learning models and pre-training language models have been widely used in the field of software classification in recent years. For example, Reference [13] proposed a multi-classification framework for Android applications based on Convolutional Neural Networks (CNN) by extracting multiple static features and combining natural language processing and deep learning. The classification accuracy of nine categories is 99.9064%. However, the dataset used in this document contains only about 5000 samples and has not been verified on large-scale datasets. The multimodal classification method of text metadata and images based on application programs was proposed in reference [14]. This approach involves the classification of images using a Region-based Convolutional Neural Network (RCNN) and integrates a text classifier to classify the applications. By doing so, it addresses the issue of low recall in classification models caused by short or incorrect software description texts, to a certain extent. Reference [15] proposed the classification of Android software based on software description text using the CRNN model, which combines CNN and RNN. It successfully classified the software into 17 categories, but it did not proceed to further subclassify each category.

2.2. Text Classification Technology

With the rapid development of natural language processing technology, increasingly more software researchers expect to learn from natural language processing technology to study this field. This work mainly classifies the application software based on the classification results of the software describing text information, so the text classification model is applied to study the software description text information. Currently, the research emphasis in text classification models primarily revolves around neural network models and transformer models. For example, Reference [16] completed emotional subclassification combining CNN and Long Short-Term Memory (LSTM) model architectures. The experimental results show that LSTM can learn long-term dependencies from more advanced representation sequences, can understand text semantic information at a deeper level, and achieve better classification results. Reference [17] made full use of the self-learning advantage of the deep learning model, introduced attention mechanism, and realized text classification by combining the Recurrent Neural Network (RNN) model to learn text features. With the advance of the pre-training language model in the field of natural language processing, Reference [18] creatively applied the pre-training language BERT model to document classification. Even though documents often contain longer sentences and multiple tags compared to typical BERT models, the model can still achieve state-of-the-art performance on four popular datasets after fine-tuning. Different fine-tuning methods of the BERT model in text classification are studied in Reference [19], and a general solution is provided for BERT fine-tuning. This scheme achieves the most advanced performance on eight widely studied text classification datasets. Reference [20] generated document embeddings using Sentence-BERT (SBERT), a variant of the pre-trained language model BERT, and then achieved topic classification by generating document topics by clustering algorithms. In Reference [21], two methods were employed to detect fake reviews. The first method involves using frequency-weighted features, which are then inputted into a machine learning classifier for classification. In contrast, the second method utilizes semantic encoding of the text data and employs a simple feed-forward neural network for classification. The experimental results clearly demonstrate that the second method is superior to the first method, and it is particularly well-suited for handling large-scale datasets. The study documented in Reference [22] presents an enhanced methodology for the BERT model by integrating it with the BiLSTM-CNN deep model to effectively classify

requirements. In Reference [23], the design and implementation of a fake news classifier using Mini-BERT is presented. The classifier further benefits from the inclusion of attention mechanisms, which help the model prioritize important vocabulary and contextual information.

As indicated in Table 1, pre-trained language models demonstrate remarkable proficiency in text classification tasks. However, there is still a slight deficiency in the research regarding software classification.

Table 1. Text classification technology.

Reference	Method and Model	Application
[13]	CNN	Android application classification
[14]	RCNN	Application classification
[15]	CRNN	Android application classification
[16]	CNN and LSTM	Fine-grained emotion subclassification
[17]	RNN	Text classification
[18]	BERT	Document classification
[19]	BERT	Different fine-tuning methods of the BERT model in text classification
[20]	SBERT	Topic classification
[21]	TF-IDF, Semantic Features	Detect deceptive and real text reviews
[22]	BERT and BiLSTM and CNN	Requirement classification
[23]	Mini-BERT	Fake news classification

2.3. Data Enhancement Technology

There is a serious data imbalance in the dataset used. In order to alleviate this problem, this work attempts to use data enhancement technology to enhance the dataset, and data enhancement is also a hot topic in text classification. For example, it is pointed out in Reference [24] that data enhancement technology uses prior knowledge and relatively simple algorithms to derive new training data from the original training data. It is an effective way to enrich the original data with task-specific knowledge. In the task of text classification, References [25,26] attempted to use the method of back-translation for data enhancement, specifically, to translate the existing example X in Language A into another Language B, and then translate it back to A to expand the dataset. It is also pointed out in Reference [27] that reverse translation can generate different interpretations while retaining the semantics of the original sentence, thus improving the performance of the model on the dataset; although reverse translation can better maintain the global semantics of sentences, however, this method cannot control which words in sentences are retained, and these words are really important to the task of topic classification because some keywords can provide more information than others in determining the topic. Therefore, it is proposed in Reference [24] to replace uninformative words with low-TF-IDF words, while retaining high-TF-IDF words for data enhancement. Reference [28] argued that common strategies for data enhancement include splitting and exchanging, adding random words, reverse translation, adding high-TF-IDF words, deleting low-TF-IDF words, replacing synonyms, and so on. At the same time, it is proved by experiments that using a replacement synonym strategy to expand training data can effectively improve the performance of the BERT model on this dataset.

2.4. BERTopic Topic Clustering Model

BERTopic is a topic clustering model proposed in 2022 [7]. Its main idea is to use a pre-trained model to embed the input text, generate a high-dimensional vector of the text, reduce the dimensionality of the high-dimensional vector, and then use the clustering algorithm to cluster the vector, to divide the text with similar characteristics into the same topic, and at the same time use the weighting scheme to calculate the weight of the keywords in each topic and extract the keywords of each topic. The model is mainly carried out by the five steps of text embeddings, dimensionality reduction, clustering, tokenizer,

and the weighting scheme, each step containing a variety of processing methods. The whole model is a modular structure, and researchers can build their topic model according to their needs, as shown in Figure 1.

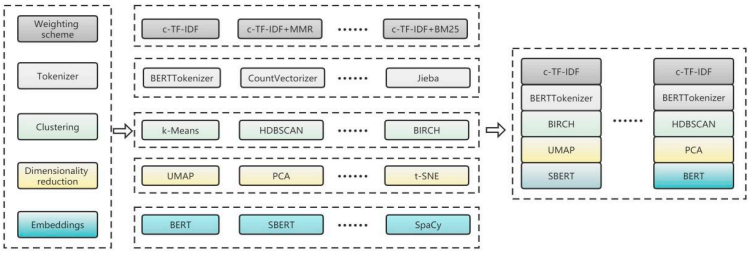

Figure 1. BERTopic topic clustering model.

The BERTopic model makes full use of the advantages of a pre-training language model in text coding in advanced natural language processing technology, which enables it to solve the common problems of phrase dependence and semantic fuzziness in traditional topic models. Its modular structure can provide researchers with a customized topic clustering model, which is mainly based on the software description text to classify the application function dimension. According to the problems existing in the software classification of the above analysis and the characteristics of the BERTopic topic clustering model, a customized topic clustering model is designed, which will be introduced in the following sections.

3. Problems and Challenges

Based on the above analysis, the problems and challenges faced by this work are as follows:

1. The construction of an application software labeling system lacks standardization and timeliness; the first challenge, therefore, is to design a method to automatically build a software labeling system to objectively and accurately assign software labels according to the semantic information of software description text, and ensure the timeliness of the software labeling system.
2. The extraction of subject words in the topic clustering model lacks refinement, resulting in the extracted subject words containing a large amount of duplicate or redundant information; the second challenge, therefore, is to design a method for refining and extracting subject words to more reasonably and effectively weigh the relevance and difference of subject words so that the obtained subject words are more interpretable.
3. There is a serious imbalance problem in the software description text data, so this is the third challenge: to design a data enhancement method to alleviate the imbalance of the data set, to improve the effect of classification.

Given the above problems and challenges, the following solutions are put forward.

1. Aiming at the first challenge, a customized topic clustering model based on BERTopic is designed, in which the software description text cluster obtained by topic clustering is used as the application software category, and the subject words extracted from the cluster are used as software tags. This method can automatically construct the software category system and fine-tune the model according to the update of the software description text to ensure the timeliness of the software label system.
2. Aiming at the second challenge, a method of multi-stage extraction of subject words based on Maximal Marginal Relevance (MRR) is proposed, which selects keywords with high correlation and high difference, iteratively, to construct a set of subject words. This method can refine and extract the representative and unique subject words to ensure the quality of the extracted subject words.
3. Aiming at the third challenge, a data enhancement method based on the c-TF-IDF algorithm is proposed, which uses this algorithm to calculate the ranking of important words in the software description text in this category, finds synonyms based on

Wordnet, and replaces important words with the same part of speech to obtain a new software description text. This method can ensure the semantic consistency of the new software description text, to alleviate the problem of serious data imbalance in the dataset.

4. Methodology

This section describes the methods presented in this article in detail. First, the overall framework of the approach will be given; then, the framework structure and implementation methods will be analyzed in detail from four perspectives: topic clustering, automatic label update, data enhancement, and classification model.

4.1. Overall Framework

Combining the idea of topic clustering and text classification, a software fine classification framework based on the software application function dimension is proposed. The overall framework is shown in Figure 2. The main objective is to achieve automatic tagging of the software description text dataset and automatic updating of the classification labels through a topic clustering model. This approach enables fine-grained software classification by utilizing a combination of pre-trained language models and neural network models. The specific formal description process is as follows:

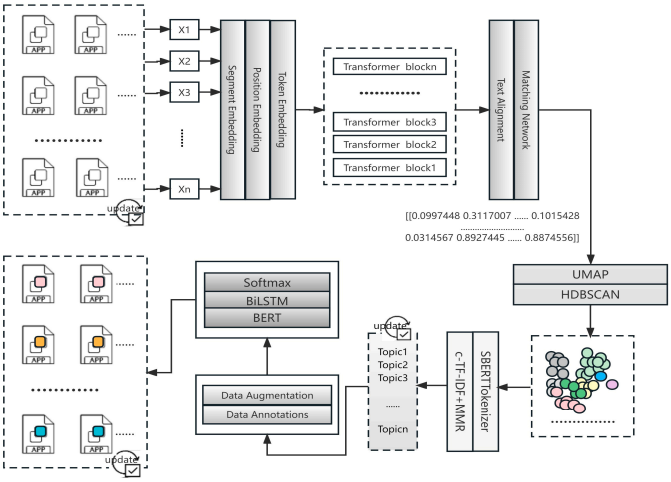

Figure 2. Model overall framework.

Input: A collection of software description text $D = \{d_1, d_2, ..., d_n\}$.

Output: clustering result C and thesaurus set T; labeled dataset L; software subdivision result collection Y.

Steps:

1. Use the pre-trained language model to embed the text as a vector to obtain a high-dimensional vector set $V = \{v_1, v_2, ..., v_n\}$;
2. Reduce the dimensionality of the set V to obtain the low-dimensional vector set $W = \{w_1, w_2, ..., w_n\}$;
3. Use the hierarchical density clustering algorithm to perform cluster analysis on the set W, and obtain the clustering result $C = \{c_1, c_2, ..., c_k\}$;
4. A multi-stage method for extracting subject words is proposed, and for each cluster c_i, the subject words are extracted as a candidate set of classification labels, and the subject term set $T = \{t_1, t_2, ..., t_m\}$ is obtained;
5. Use the subject words to label the software description text to obtain the labeled software description text dataset, expressed as $L = \{(d_i, t_i) | d_i \in D, t_i \in T\}$;
6. For the newly added software description text, iteratively update its software label;
7. A new data augmentation technique is proposed to enhance the data of dataset L to reduce the imbalance of the dataset;

8. Based on the dataset, the classification model is trained and tested, and the result set of the software subdivision class $Y = \{y_1, y_2, ..., y_n\}$ is obtained.

4.2. Topic Clustering

It is expected that the clustering results of the software description text and the collection of subject words will be obtained through the topic clustering model, to realize the automatic tagging of the software label and obtain the tagged software description text dataset. Based on the problems existing in the object software description text, such as fuzzy semantic description, different description forms, lack of description information, subjectivity, and bias in the description, combined with the research goal of the topic clustering stage, a customized BERTopic model is designed, and the model structure is shown in Figure 3.

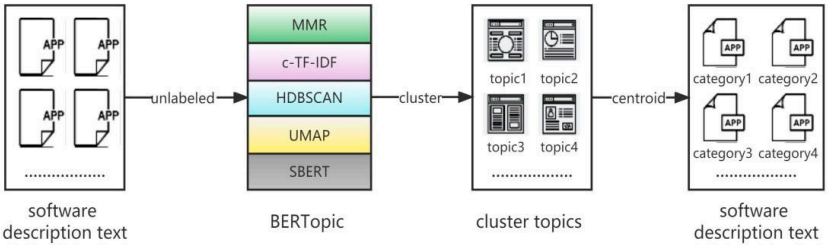

Figure 3. BERTopic model.

4.2.1. Embed Documents

The constructed topic clustering model uses SBERT [29] to generate the embedded vector of the software description text and represents the software description text as points in the high-dimensional vector space, which is ready for subsequent topic clustering. The process of embedded representation of software description text using the SBERT model is described as follows:

1. Input: Software description text A: $A = [s_1^A, s_2^A, ..., s_n^A]$. Software description text B: $B = [s_1^B, s_2^B, ..., s_m^B]$. Where s_i^A and s_j^B represent sentences in the text, and n and m represent the number of sentences in both texts, respectively.
2. Generation of text embedding vector: For each sentence in the text, the pre-trained SBERT model is used to transform the sentence into an embedded vector. The embedding of software description text A is represented as: $E_A = [e_1^A, e_2^A, ..., e_n^A]$. The embedding of software description text B is represented as: $E_B = [e_1^B, e_2^B, ..., e_m^B]$. where, e_i^A and e_j^B represent the embedding vector of the corresponding sentence.
3. Similarity calculation: Corresponding to any two sentences in the software description text, the cosine similarity is used to measure the similarity between sentences. The similarity scores of the i sentence in the software description text A and the j sentence in the software description text B are as follows: $S_{i,j} = cosine_similarity(e_i^A, e_j^B)$, cosine_similarity(.) represents the cosine similarity calculation function.

What need to be emphasized in the whole embedded representation process are the two key stages of the model, the Text Alignment and Matching Network. In the Text Alignment stage, the attention weight between two sentences in the description text of the software is calculated to determine the alignment position of each word in another sentence and then aligned with the embedded representation of the sentence according to the alignment position, and the embedded representation after alignment is obtained. In the Matching Network stage, the features of the aligned sentence embedding representation are extracted and activated, then the similarity score between the texts is calculated based on the feature representation, and finally the embedded vector representation of the software description text for the subsequent topic clustering task is obtained.

4.2.2. Dimensionality Reduction

The high-dimensional software description text vector is obtained in the above steps. Before clustering, the model framework uses the Uniform Manifold Approximation and Projection (UMAP) [30] dimensionality reduction algorithm to reduce dimensionality. The specific description process is as follows:

Suppose the above-embedded vector is stored in the matrix X, $X \in R^{\wedge}(N \times M)$, where N is the number of software description texts, and M represents the original dimension of the software description texts.

1. Calculate the distance matrix D: Calculate the similarity between the software description texts, and obtain the distance matrix $D \in R^{\wedge}(N \times N)$. The element of the distance matrix $D_{i,j}$ represents the distance or similarity between the i-th software description text and the j-th text.
2. Construct local neighborhood: for each software description text, select its k-nearest neighbors as its local neighborhood, and find the k-nearest neighbors of each text based on the distance matrix D.
3. Define joint probability distribution: using local neighborhood information, calculate the joint probability distribution $P_{i,j}$ between each pair of software description texts, where $P_{i,j}$ represents the similarity between the i-th text and its adjacent text j.
4. Optimize low-dimensional representation: Use the UMAP algorithm to map high-dimensional embedded vector representation to low-dimensional space. Minimize the difference between the joint probability distribution $P_{i,j}$ between high-dimensional data points and the symmetric joint probability distribution $Q_{i,j}$ between low-dimensional data points to obtain a low-dimensional embedding representation $Y \in R^{\wedge}(N \times d)$, where d $(d \leq M)$ is the dimension of the target after dimensionality reduction.

Finally, we can obtain the embedded vector representation Y of the software description text in the reduced-dimensional low-dimensional space, and each text is represented by a vector. These vectors are stacked in a matrix in the order of the text, that is, $Y \in R^{\wedge}(N \times d)$.

4.2.3. Clustering

After the low-dimensional vector representation Y of the software description text is obtained, the model framework uses the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [31] algorithm for clustering. The expression process of the algorithm for clustering the software description text is as follows:

1. Calculate the distance matrix: calculate the distance matrix $D^{'}$ between the software description texts according to the low-dimensional vector representation Y.
2. Set clustering parameters: Set clustering parameters before using the HDBSCAN algorithm. Including the minimum number of samples (min_samples) and neighborhood distance threshold (min_cluster_size), these parameters will affect the clustering results.
3. Build a cluster diagram: Use the distance matrix D′ and clustering parameters to build a cluster diagram. A cluster graph refers to the tree structure connecting data points, in which the leaf node represents a single data point, the internal node represents the cluster, and the edge represents the relationship between the clusters.
4. Extract the clustering results: Extract the specific clustering results by analyzing the cluster graph. Based on the connection degree and density information in the cluster diagram, the cluster to which each software description text belongs is determined. At the same time, we can also identify the noise points, that is, the data points that do not belong to any clustering, and finally obtain the topic clustering result C.

It should be noted that the HDBSCAN algorithm is a density clustering algorithm, which does not need to specify the number of clusters in advance. The algorithm can automatically identify the number and shape of clusters, which meets the requirements of this work for clustering software description text datasets that do not specify the number of software categories.

4.2.4. Create Topic Representation

After obtaining the clustering results of the software description text, the c-TF-IDF and MMR algorithms [32] are used to extract the subject words, which can represent the key features of each cluster. To obtain keywords closer to software tags, this work introduces the methods of deactivated word processing and multi-stage keyword extraction to optimize the subject word extraction method; the specific description process is as follows:

1. Stop word processing: for each cluster of software description texts $C = \{d_1, d_2, ..., d_n\}$, use the stop word list to filter out common words such as prepositions, conjunctions, etc.
2. c-TF-IDF weight calculation: For each term t in cluster C, calculate its c-TF-IDF weight $score(t, c)$which reflects the importance of term t in cluster C, as well as the conceptual relevance in the entire cluster C. The c-TF-IDF weight $score(t, c)$ is calculated as follows:

$$score(t, c) = TF(t, c) \times \log(1 + \frac{N}{DF(t)}). \quad (1)$$

It is calculated by taking the logarithm of the average number of words per cluster N divided by the frequency of term t across all clusters. $TF(t, c)$ represents the word frequency of the word item t in cluster C.

3. Initial topic word selection: for each cluster C, k word items with the highest c-TF-IDF weight are selected as the initial subject word set $T(d)$.
4. The MMR iteration process: repeat the following steps until the stop condition is met:

(1) For each cluster C, calculate the MMR value of each non-subject term t and the selected subject word set $T(d)$, and the MMR value is calculated as follows:

$$MMR(t, T(d)) = \lambda \times score(t, c) - (1 - \lambda) \times MAX(MMR(t^{'}, T(d))). \quad (2)$$

where λ represents the parameter that weighs the degree of correlation and difference, and t' represents the selected subject word.

(2) Select the non-topic word item t with the highest MMR value and add it to the subject word set $T(d)$ of the cluster C.

(3) For each cluster C, update the c-TF-IDF weight to reflect the new set of subject words $T(d)$.

5. Final topic word selection: merge the subject word set of all documents into the final subject word set T.

It should be noted that in the process of MMR iteration, different stop conditions can be set to determine when to end the iteration, such as reaching the specified number of iterations, the stability of the set of subject words during the iteration, the fixed size of the set of subject words, and the change or threshold of MMR values, etc., which can be tried separately in the course of the experiment to choose the best stop condition. After obtaining the subject words of each cluster, several of the most representative topic words will be selected as the topic tags of the cluster. For each piece of software description text, the subject label of the corresponding cluster is assigned to the text as a classification label according to the cluster to which it belongs, and the labeled dataset L is obtained.

4.3. Automatic Label Update

With the continuous updating of the software description text, the trained BERTopic model can dynamically update the topic clustering results to realize the automatic update of the software label. The specific methods are as follows.

1. Input: Original software description text data collection: $D_{old} = \{d_{old1}, d_{old2}, ...d_{oldn}\}$. New software description text data: $D_{new} = \{d_{new1}, d_{new2}, ...d_{newn}\}$.
2. Combine text data to obtain the overall software description text data collection: $D = D_{old} \cup D_{new}$

3. Train or fine-tune the BERTopic model: use dataset D to train or fine-tune the original BERTopic model M: $M^{'} = M.train(D)$.
4. Cluster text data: cluster dataset D using the updated BERTopic model M': $C = M^{'}.transform(D)$, where the clustering result C is obtained, where each text d_i is associated with the corresponding cluster-ID.
5. Update the software classification label: extract the keyword for each cluster C_j : $K_j = extract_keywords(C_j)$, use the keyword K_j as the subject word of the cluster C_j, and select the subject word as its classification label according to the subject word collection of the cluster to which each software description text d_i belongs.
6. Output: updated cluster: $C = \{c_1, c_2, ..., c_k\}$, updated software classification label: $L_{new} = \{l_{new1}, l_{new2}, ..., l_{newn}\}$, where L_{newi} represents the classification label of the text d_i.

The above process enables the updating of clustering topics and software classification labels to maintain consistency with the text data described by the latest software. It is worth noting that if the software description text is assigned to the new cluster during the topic update, a new label needs to be generated, If the software description text is assigned to the original cluster, it may be possible to use directly the existing tags associated with the cluster for classification.

4.4. Data Enhancement

The labeled dataset exhibits a significant data imbalance, whereby certain subclasses contain only a few dozen software instances while others consist of thousands. This dataset imbalance critically impacts the efficacy of the subsequent classification module. Accordingly, a new method of data enhancement is proposed. The importance $W_{t,c}$ of each word in each software description information text is calculated using the c-TF-IDF algorithm mentioned above, and sorted according to $W_{t,c}$ to form the important word set W of each software information description text. At the same time, the stop words and words with a low order of importance are filtered, and then the synonyms of the words in the set W are found in turn, and the synonyms are replaced to form a new software description information text. Based on WordNet to find synonyms, the replacement candidate word set W_{rep} of important word set W is established, the synonyms with the same part of speech are selected from the replacement candidate word set to replace them, and the new software information description text is generated using the replaced synonyms. The dataset is expanded on the premise of ensuring the semantic similarity of the text. The algorithm is described in detail, as Algorithm 1:

Algorithm 1 Data enhancement algorithm based on c-TF-IDF

Input: $X = \{W_1, W_2, ..., W_n\}$, subclass dataset, WordNet
Output: Generate new sentence data: subclass dataset L

For each word W_t in X do
  Compute the $W_{t,c}$ Via c-TF-IDF
end for
Create a set W of all $W_t \in X$ sorted by the descending order of $W_{t,c}$
Filter W
For each word $W_t \in W$ **do**
  Find synonyms using WordNet
  Create a set $R = \{W_1, W_2, ..., W_{rep}\}$ of replacement candidate words
  Search the set R in order agreement of speech (W_t, W_{rep})
  If speech (W_t, W_{rep}) **then**
  Replace $W_t$with $W_{rep} \in R$
  Delete $W_{rep} \in R$
  Create a set $X' = \{W_1, W_2, ..., W_{rep}\}$
  end if
end for

4.5. Classification Model

The model used in the classification module is a deep learning model integrated by BERT and BiLSTM; BERT is adopted to generate context-dependent word vectors and use them as input to BiLSTM. Finally, the Softmax function is used to predict the software fine classification tags. The structure of the model is shown in Figure 4, which mainly consists of the BERT word embedding layer and BiLSTM network-layer decoding layer. The word embedding layer uses the pre-training model BERT to transform the input sequence into the corresponding word vector sequence, and the BiLSTM layer uses the forward network and backward network to encode the current context information to capture the long-sequence dependencies in the BERT model-output software description text vector sequence. The decoding layer processes the output of BiLSTM nonlinearly through multiple full connection layers, maps it to the spatial domain of the classification result, and uses the Softmax function to output the probability of each category.

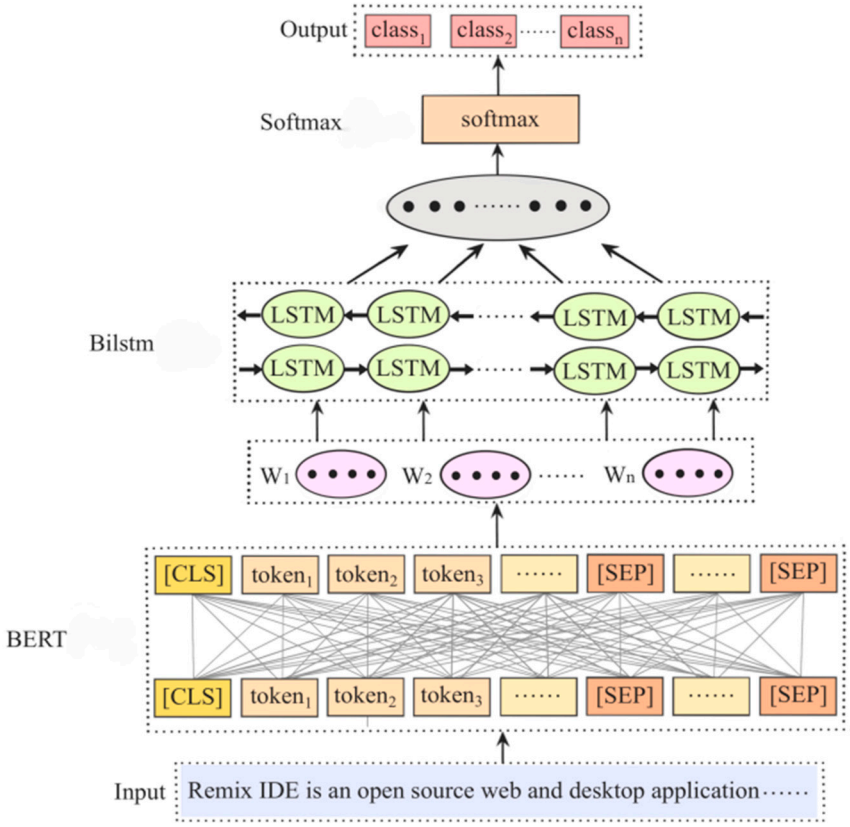

Figure 4. BERT-BiLSTM model.

5. Experiment and Evaluation

In this section, we will validate the effectiveness of the proposed method through clustering and classification experiments on two datasets. We will then compare and analyze the experimental results to draw meaningful conclusions.

5.1. Dataset

To verify the method proposed in this work, the application text description information was crawled from international popular application stores such as SourceForge, 360 Software Housekeeping, Huawei App Store, and Mi Store, and 21 categories were crawled from the SourceForge store, with a total of 22,5903 software description text messages as the original dataset 1. The text description information of the application software was crawled from several application stores in China, and the categories were merged and adjusted. Finally, the data were identified as belonging to 19 categories, with a total of

405,910 pieces of software description text information as the original dataset 2. Table 2 presents an overview of the datasets.

Table 2. Dataset.

Dataset	Level-1 Categories	Number	Count	Dataset	Level-1 Categories	Number	Count
Dataset1	Blockchain	0	349	Dataset2	Programming development	0	2114
	Communications	1	15,827		Multimedia class	1	18,791
	Database	2	4763		Education and teaching	2	12,481
	Desktop environment	3	5429		Chat communication	3	11,358
	Education	4	4666		Graphic images	4	22,677
	Formats and protocols	5	5631		Network software	5	35,320
	Games entertainment	6	21,361		System tools	6	54,835
	Internet	7	24,439		Industry software	7	4158
	Mobile	8	1280		Utilities	8	45,195
	Multimedia	9	19,949		Recreation	9	55,771
	Office business	10	8736		Thematic software	10	2400
	Other non-listed topics	11	4060		Security antivirus	11	6129
	Printing	12	751		MAC software	12	8846
	Religion and Philosophy	13	506		MAC gaming	13	357
	Scientific Engineering	14	27,427		Android software	14	80,479
	Security	15	5839		Android games	15	30,694
	Social Sciences	16	597		Apple software	16	11,151
	Software development	17	38,291		Apple games	17	628
	System	18	30,600				
	Terminals	19	1207		Mobile phone software	18	2526
	Text editors	20	4195				

As shown in Table 2, the categories in the two datasets are numbered from 0 as category identifiers, and the software description text information contained in each category is counted.

5.2. Experimental Environment

The configuration of the lab environment is shown in Table 3.

Table 3. Experimental environment.

Platforms	Content
Hardware dependencies	Nvidia RTX 2060 graphics card, Ryzen R7-4800h processor
Software dependencies	Python3.9 TensorfFlow 2.10.1 Scikit-learn 1.0.2 Numpy 1.12.5 Pandas 1.5.2 Matplotlib 3.4.3
GPU components	NVIDIA GPU GeForce RTX2060 6GB Cuda 11.4 Cudnn 8.2.4

5.3. Analysis of the Results of Clustering Experiment

Clustering experiments were carried out on two datasets, and three graduate students of our team were invited to manually analyze and verify the clustering results. Taking the clustering experiment on dataset 1 as an example, the clustering experiment and the process of constructing subclass tags are analyzed and explained. Dataset 1 contains a total of 21 categories, of which 11 categories are divided into subclasses, with a minimum of 2 subclasses and a maximum of 26 subclasses. In this experiment, topic clustering was carried out on the categories containing subclasses in dataset 1, and three graduate

students were asked to compare the clustering clusters and subject words with the original subclass tags in the dataset. The quality of tags was obtained from the size of clusters, the correlation between the same cluster of subject words, readability, and the dissimilarity verification of topic words between different clusters. Additionally, the number and labels of the original subclasses are compared to judge the effect of the topic clustering results. Then, the categories without subclasses are clustered, and the optimal clustering results are obtained by constantly adjusting the clustering parameters. Furthermore, the number of subclasses and subclass tags in this category are determined by combining them in a manual analysis, and a complete subclass label system is constructed.

Taking the Communications category in dataset 1 as an example, which contains 12 subclasses, 15,827 software units describe text information and the subclass tags are removed for clustering experiments. If the default parameters are used, the clustering algorithm utilizes a minimum number of samples, denoted as min_samples (minimum sample value), set to 2, and a neighborhood distance threshold, denoted as min_cluster_size (neighborhood distance threshold), set to 0.75. The number of topics obtained by clustering is 217, and the range of topic sizes varies from 11 to 522. A visual topic graph of the output from the clustering results was analyzed, as shown in Figure 4. The size of the circle in the diagram represents the scale of the topic. Obviously, the topics obtained by clustering are too trivial and the distances between a large number of topics are very close, which means that there is a large similarity between topics, so this line of work tries to optimize by adjusting the clustering parameters.

According to the continuous adjustment of the clustering parameters, it was determined that the clustering results were better when there were 50 and 20 clusters, respectively. The visualization of the topics obtained by the experiment is shown in the middle and right of Figure 5. The red circle in the figure represents the currently active theme. According to the visualization, the theme size is relatively average. The size of the topics in the middle of Figure 5 ranges from [631–2008], while on the right-hand side of Figure 5 it ranges from [994–2348]; in terms of topic distance, however, topics that can be merged still exist.

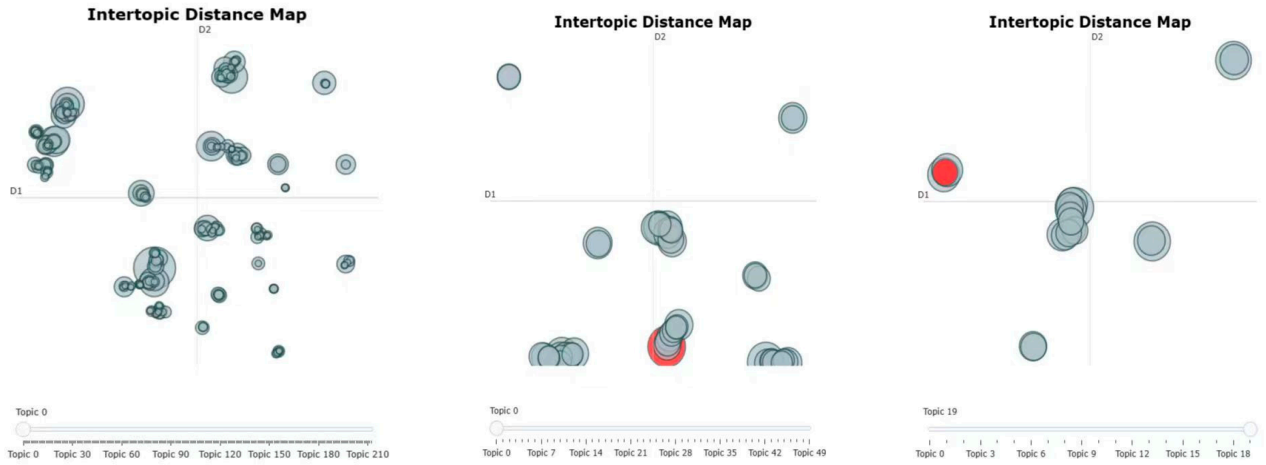

Figure 5. Topic visualization.

For a more detailed analysis of the results of a clustering experiment, the topic representation and the hierarchy between topics are observed from the topic hierarchy graph output in the experiment; the topic hierarchy diagram is shown in Figures 6 and 7. The left-hand side shows the topics obtained by clustering: the topic representation is composed of the top four topic keywords with the highest score, and the hierarchy on the right shows the hierarchical relationship between the topics obtained by the clustering, according to which the hierarchical merging between topics can be carried out.

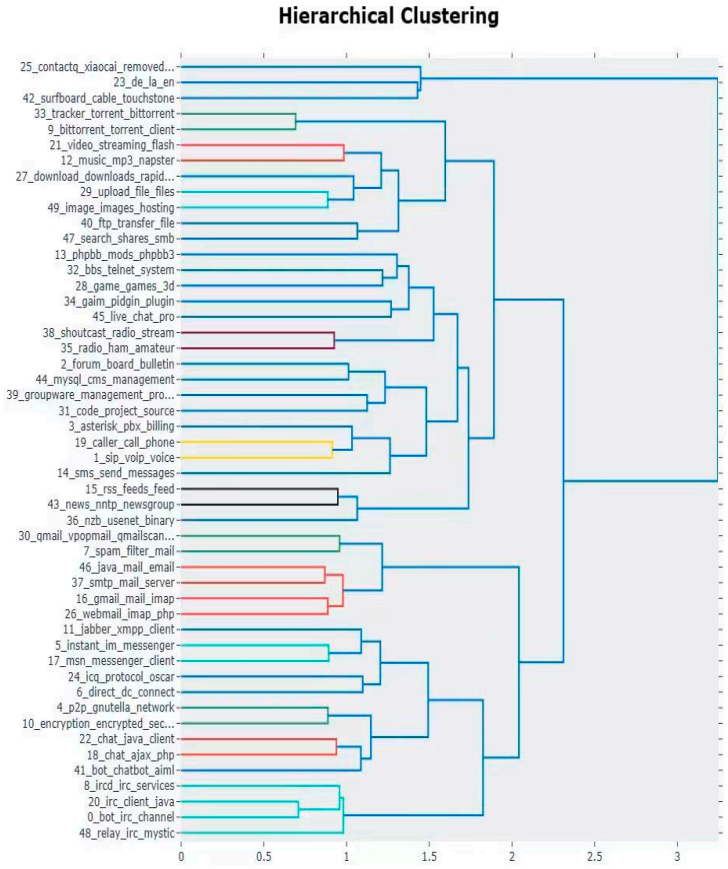

Figure 6. Topic hierarchical structure diagram 1.

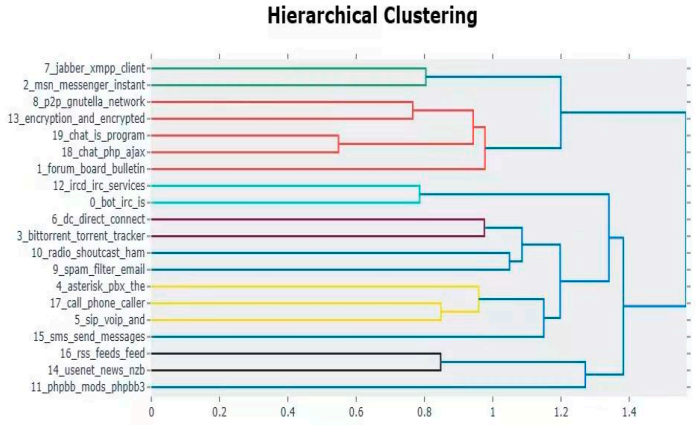

Figure 7. Topic hierarchical structure diagram 2.

To compare and analyze the relationship between the topic words extracted in the topic clustering experiment and the original subclass tags, the first 20 topic representations in the clustering experiment (sorted by the number of samples contained in each topic) are shown in Table 4, from which it can be seen that in the two experiments with topic clustering results of 50 and 20 respectively, 8 of the first 10 topics are consistent, indicating that the evaluation indicators of the clustering topic model are consistent. Compared with the original tags, the software description texts in which the clustering model is clustered into 0_cms_content_management_syste,1_game_space_strategy_the,8_forum_phpbb_board _bulletin, and other topics appear in a large number of BBS subclasses. After analysis, it is proved that the reason why the software description text of 1_game_space_strategy_the theme appears in the BBS subclass is that a large number of online games that interact based on forums in this category are assigned to BBS tags in the application market, and

the subclass tags with more appropriate software description texts should be online games. For the other two topics, 0_cms and 8_forum, compared with BBS tags, the classification granularity is finer, indicating that another evaluation index of the clustering topic model has better topic diversity.

Table 4. Comparison of two clustering experiment topics.

Original Subclass Label	Subject Representation	
	Second Clustering Experiment	**Third Clustering Experiment**
BBS	0_cms_content_management_system	0_mail_email_to_spam
Chat	1_game_space_strategy_the	1_cms_content_management_system
Conferencing	2_de_para_la_en	2_mp3_music_files_player
Email	3_irc_bot_bots_is	3_game_the_space_and
File-sharing	4_bittorrent_torrent_p2p_client	4_emulator_the_for_and
File-sync	5_video_dvd_ffmpeg_mplayer	5_irc_bot_to_is
Ham-radio	6_search_engine_samba_index	6_de_para_la_en
Internet-phone	7_traffic_network_packet_packets	7_video_dvd_media_ffmpeg
Rss-feed-readers	8_forum_phpbb_board_bulletin	8_editor_text_latex_is
Streaming	9_mp3_tags_tag_files	9_forum_phpbb_board_bulletin
Telephony	10_encryption_encrypted_encrypt_key	10_search_to_web_the
Usenet-news	11_rss_news_feeds_feed	11_accounting_and_financial_money
	12_pdf_documents_files_to	12_network_traffic_snmp_monitoring
	13_emulator_terminal_the_emulation	13_backup_backups_to_files
	14_sequence_genome_sequencing_of	14_test_testing_tests_unit
Not labeled	15_notes_todo_task_note	15_image_images_metadata_and
	16_asterisk_sip_voip_pbx	16_network_packet_snort_tcp
	17_framework_php_mvc_ajax	17_calendar_outlook_date_to
	18_package_packages_slackware_gentoo	18_xml_parser_grammar_and
	19_mail_email_spam_smtp	19_engine_3d_game_opengl

The above results also confirm that the clustering of the model in the software text description information is effective, and can guide the division of subclasses and the formulation of labels in the application software.

On this basis, three graduate students of our team arranged to manually label, according to the number of topics clustered, topic scale, topic keywords, and software description text. Finally, the complete label system of dataset 1 was obtained, and some of the label systems are shown in Table 5. The blue font labels in Table 5 represent new themes that use the theme update method proposed earlier.

Table 5. Subclass label.

Category Labels	Subclass Labels	Category Labels	Subclass Label
Blockchain	Technology platform Data storage platform Payment system Block explorer	Database	Engines and Servers Backup and Recovery Performance optimization Front and Ends
Communications	Instant messaging tools Forum and Chat providers Remote collaboration Social network CMS Feed aggregation File sharing and hosting E-mail	Desktop environment	File management Text editing Image processing Audio and video playback Web browsing Office software Password management Notepad
	Rss-feed-readers Video and speech Virtual worlds	Education	Computer-aided Exam Educational management
Formats and protocols	Data formats Protocols		Translation dictionary Online classroom

5.4. Analysis of the Results of Classified Experiments

5.4.1. Classified Evaluation Index

Common evaluation metrics in classification models are Accuracy, Precision, Recall, and F1 Score. In addition to the above indicators, the multi-classification model also includes macro AVG, micro AVG, and weighted AVG. According to the above dataset analysis, there are many classification categories in this work dataset, and there is a data imbalance between different categories, so the Precision, Recall, and F1-Score evaluation indicators of macro average and weighted average are employed to evaluate the model, which will help to evaluate the classification model used in this work more effectively.

5.4.2. Data Augmentation Validity Evaluation

The data enhancement method proposed above is used to enhance the labeled dataset. The data enhancement processing is carried out for the subclasses in which the samples in the subclasses are lower than the average values of all samples in the subclasses, and the effects of data enhancement on the classification effect of the classification model are compared and analyzed. In this experiment, utility software in dataset 2 is utilized for verification and analysis. The main reasons for the experiment are: first, the large amount of data in the large dataset has 45,195 pieces of software text information; second, the number of subclasses is 16 subclasses; third, there is a serious imbalance in the data. Some subclasses have only 690 samples, and some subclasses have as many as 9640 samples, which significantly affects the effect of the classification model.

In this experiment, the data of subclasses lower than the mean of the subclass sample of 2824 were enhanced sequentially, and the classification effect of the classification model was verified. From Table 6, it can be seen that under the original dataset, the macro-average Precision, Recall, and F1 Score of the classification model are 0.89, 0.87, and 0.88 respectively, and the weighted average Precision, Recall, and F1 Score are 0.89, 0.89, and 0.89 respectively. Meanwhile, under the enhanced dataset, the macro-average Precision, Recall and F1 Score of the classification model are 0.91, 0.92, and 0.91, respectively, and the weighted average Precision, Recall and F1 Score are 0.90, 0.91, and 0.91, respectively.

Table 6. Data-enhancement validity experiment.

Subclass	Count		Precision		Recall		F1 Score	
	Original Data	Data Enhancement	Original Data	Data Enhancement	Original Data	Data Enhancement	Original Data	Data Enhancement
office software	1080	2160	0.88	0.92	0.64	0.88	0.74	0.90
editing tools	790	1580	0.89	0.90	0.84	0.92	0.86	0.91
printing tools	6770	6770	1.00	0.99	0.99	0.99	0.99	0.99
e-reader	2050	4100	0.97	0.98	0.98	0.98	0.97	0.98
management tools	2320	4640	0.94	0.95	0.94	0.96	0.94	0.95
calculating devices	1160	2320	0.75	0.88	0.77	0.90	0.76	0.89
keyboard and mouse	3260	3260	0.72	0.72	0.82	0.82	0.77	0.77
calendar clock	690	1380	0.91	0.94	0.90	0.92	0.91	0.93
input method	9640	9640	0.88	0.82	0.86	0.85	0.87	0.83
file processing	1880	3760	0.95	0.94	0.73	0.86	0.83	0.90
file management	1860	3720	0.73	0.84	0.77	0.86	0.75	0.85
compression and decompression	5390	5390	0.89	0.88	0.91	0.91	0.90	0.89
miscellany	1510	3020	0.92	0.95	0.93	0.94	0.92	0.94
conversion translator	4650	4650	0.89	0.89	0.94	0.95	0.91	0.92
font tools	965	1930	0.97	0.98	0.94	0.96	0.96	0.97
font download	1180	2360	0.91	0.92	0.97	0.97	0.94	0.94
macro avg			0.89	0.91	0.87	0.92	0.88	0.91
weighted avg			0.89	0.90	0.89	0.91	0.89	0.91

The above experimental results suggest that the proposed data enhancement method can effectively alleviate the performance degradation of the classification model caused by data imbalance. In this experiment, we continue to enhance the data of the subclasses in the dataset which are lower than 2824 of the sample average of the original dataset, and find that the performance of the classification model does not continue to improve significantly,

but that the Precision has declined in some subclasses. After repeated verification, it is determined that 1–2-times data enhancement in the dataset will effectively improve the performance of the classification model, and when the number of enhancements is more than 2 times, the performance of the classification model will decline. After analysis, it is found that it is due to the decrease in text semantic similarity after many times of synonym substitution.

5.4.3. Performance Evaluation Experiments

The two-stage classification method is used for software subdivision. In the first stage, only the large classes in the dataset are classified, and the second stage is classified for the subclasses within the large classes. These two stages were experimentally verified. The data-enhanced dataset is divided 9:1, with 90% of the training set and 10% of the test set. Four kinds of experiments are carried out in two datasets: the first is the one-stage experimental verification of the overall dataset; the second is the experimental verification of the large class with the largest number of subclasses, such as the "software development" category in dataset 1, which contains 26 subclasses, and the "system tools" category in dataset 2, which contains 30 subclasses. The third is the experimental verification of the large class with the least number of subclasses, such as the "blockchain" category in dataset 1, which contains only 3 subclasses, and the "programming development" category in dataset 2, which contains 5 subclasses; the fourth is for the categories with weak differentiation between subclasses, such as the Mac game (including 8 subclasses), the Apple game (including 11 subclasses), the Android game (including 10 subclasses), leisure and entertainment (including 12 subclasses), etc.; these three kinds of experiments belong to the second stage of classified experiments. Based on these four kinds of experiments, the effect of the classification model is evaluated respectively, and the classification effect is shown in Table 7.

Table 7. Performance evaluation experiment of classification model.

Categories	Macro Avg			Weighted Avg		
	Precision	**Recall**	**F1 Score**	**Precision**	**Recall**	**F1 Score**
Software development	0.93	0.90	0.90	0.93	0.92	0.93
System tools	0.90	0.89	0.88	0.91	0.91	0.90
Blockchain	0.90	0.95	0.91	0.96	0.96	0.96
Programming-development	0.94	0.97	0.96	0.95	0.96	0.95
Mac game	0.90	0.88	0.89	0.92	0.89	0.91
Apple game	0.89	0.87	0.86	0.91	0.90	0.91
Andriod game	0.86	0.86	0.87	0.89	0.91	0.90
Leisure and entertainment	0.88	0.89	0.89	0.90	0.89	0.89
All Categories of dataset1	0.90	0.88	0.90	0.91	0.90	0.91
All Categories of dataset2	0.91	0.89	0.90	0.92	0.91	0.92

It is obvious that the macro-F1 value of this classification model on dataset 1 and dataset 2 is 0.89 and 0.90, respectively, while the weighted-F1 value can reach 0.91 and 0.92. This is mainly because the number of individual large categories in the dataset is too small, such as Blockchain, Religion and Philosophy and Social sciences in dataset 1. After data enhancement, the gap between the data scale of these categories and that of Software Development, System, and other large categories remains about 20–30 times, resulting in a low recall rate of the model in this category, which undoubtedly affects the classification effect of the classification model.

In the second phase of the classification experiment, it was observed that the performance of the "software development" and "system tools" categories, which have a larger number of subclasses, outperformed the "blockchain" and "programming development" categories with fewer subclasses. The macro-F1 value reached as high as 0.95. On the

other hand, in the category with the largest number of subclasses, the lowest macro-F1 value achieved was 0.89, indicating that this model has some advantages in software fine classification.

In the Mac game, Apple game, Android game, and leisure and entertainment categories, the classification effectiveness of this model has declined, which may be mainly attributed to the following reasons: first, some game subclasses in the dataset are not marked accurately, for example, some action shooting games are marked as combat games; second, the semantic description of the software description text information in each game subclass is not accurate enough, which reduces the accuracy of the model in classification. Third, most games in the game have multiple labels, and the model hard-labels them as a label category, which undoubtedly affects the effect of the model when classifying.

5.4.4. Model Contrast Experiment

The three deep learning models often used for text classification were selected as the benchmark models by BERT, LSTM, and textCNN, and the classical machine learning algorithms and deep learning models in this research field were selected for comparative experiments, such as the Naive Bayes classifier used in Reference [9] and the CRNN model based on the deep learning model CNN and RNN used in Reference [15]. The experimental results are depicted in Figure 8. The weighted F1 values obtained by the BERT model on the two datasets are reported as 0.89 and 0.90, respectively. It is noteworthy that the highest weighted-F1 value achieved across both datasets is 0.89, indicating a relatively favorable classification performance. However, the textCNN model exhibits a slightly lower classification effect in comparison. On the other hand, the machine learning algorithm demonstrates the poorest performance, with weighted-F1 values of 0.84 and 0.82 on the two datasets, respectively. The proposed BERT-BiLSTM model outperforms other models, showcasing superior results. The weighted-F1 values attained for this model on the two datasets are reported as 0.91 and 0.92, respectively. The accuracy and recall scores also surpass the threshold of 0.90, suggesting a high level of precision in identifying all categories with minimal instances of misclassification.



Figure 8. Model contrast experiment.

From these experimental findings, it can be inferred that machine learning models exhibit a relatively limited capacity for automatically extracting intricate textual features and capturing deep-level semantic context compared to deep learning models. Nonetheless, the application of pre-trained language models has proven to yield remarkable performance improvements.

5.5. Experimental Validation of Model Effectiveness

In order to verify the effectiveness of the proposed method, this section of the experiment will use the proposed method to experiment and verify the research subjects in Reference [15]. The experimental dataset in this literature is 17 classes with a total of 560,000 data, which describes the subcategories of the categories, but only realizes the classification of 17 categories, and does not subdivide the subcategories. This section of experiments will verify the effectiveness of the proposed method in this dataset. The experimental results are shown in Figure 9. According to the experimental results, the proposed method can divide the categories in the dataset into more fine-grained subcategories through cluster analysis; for example, this method clusters game categories into 12 subcategories such as adventure puzzle, shooting games, action games, online games, and board games, expanding upon the original dataset which only had 5 subcategories like enlightenment puzzle solving and gunfight shooting. Reference [15] focused on conducting classification experiments on the main categories within the dataset, without delving into the subcategories. The proposed method in this paper enables fine-grained classification; in the two-stage subdivision, the weighted-F1 value of the game category of this model can reach 0.88, and the weighted-F1 value of the social and communication category can reach 0.94, The effectiveness of the proposed method was confirmed.

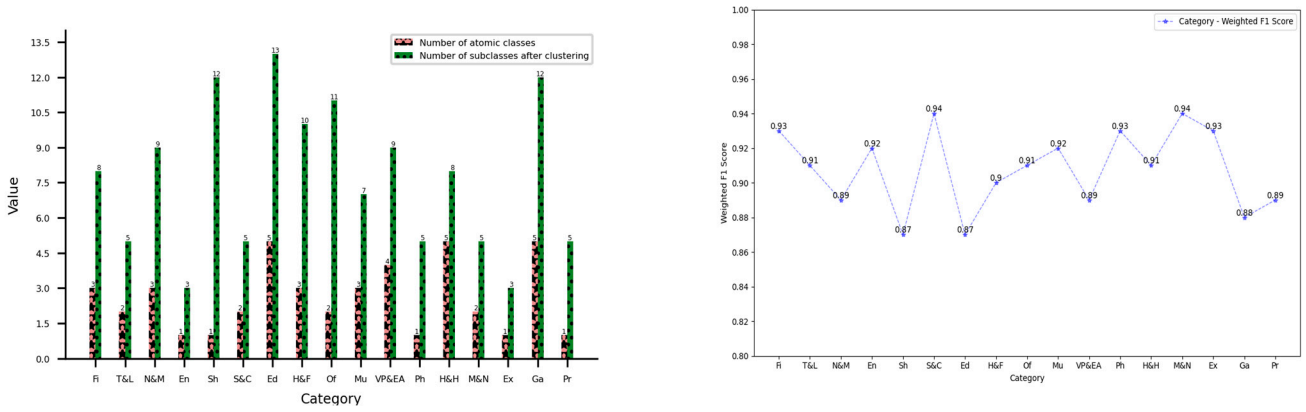

Figure 9. Experimental validation of model effectiveness.

6. Discussion

In this section, we will primarily analyze the limitations of this approach and discuss future work.

6.1. Limitations

While the experimental results have shown the method's advantages in automatic software tagging, automatic software label updating, and fine-grained software classification, further discussions and analysis indicate that there is still potential for improvement in this approach. For example, this method only subdivides the functional dimension of software from the level of software description text, so it needs to ensure the correctness of software description text, to a large extent. Once the application software lacks software description text or the software description text does not describe the software function incorrectly, the software cannot be classified or misclassified. Secondly, for software with multiple functions, it is still necessary to study further whether the core functions can be determined based on the software description text, and then assign them to the most appropriate subcategories. Finally, this method can automatically build the software labeling system, but the intermediate process still requires manual calibration. This suggests that how to automatically generate the software label system throughout the process is also a problem to be paid attention to in the future.

6.2. Outlooks

With the development of technology, future research on the fine classification of software tags may explore and adopt more advanced models and methods, such as the concept proposed in Reference [33], which can be leveraged to further enhance the generation capability of large-scale pre-training language models like ChatGPT, enabling the automatic generation of fine-grained classification label systems for software or adopting the latest technologies in the fields of natural language processing, machine learning, and deep learning to improve the performance of the application software tag fine classification model. Perhaps the expertise in the field of professional software may be integrated into the training model, so that the classification model has the ability to better understand and deal with the functions and features of the software. In addition to text information, one can also consider combining other modal information (such as images, audio, etc.) with software description text for software label subdivision, and provide more comprehensive and accurate feature representation by fusing multiple information sources, thereby improving classification results. In short, the accumulation of data and the progress of technology will enable future software label classification to achieve higher accuracy and efficiency and provide increasingly accurate support for software recommendation and management in the application market.

7. Conclusions

We propose a topic clustering model based on customized BERTopic to cluster the description texts of application software. The automatic processing of application software tags is achieved by leveraging the results of topic clustering and extracted subject words, providing a guiding method for the application store to construct the application software tag system. Furthermore, we introduce a multi-stage extraction method of subject words based on MRR to enhance the quality of subject word generation. Additionally, we propose a data enhancement method based on the c-TF-IDF algorithm to address the issue of imbalanced datasets.

The experimental results indicate that the proposed method exhibits certain advantages. However, there is still room for improvement in the classification performance of this method, particularly in subclasses with severe sample imbalances, application software with poor-quality textual descriptions, or applications with multiple functionalities. This limitation arises from solely conducting fine-grained classification of application software based on the functional dimension derived from software description text without considering the inherent functionality of the software itself. Therefore, future research should explore integrating string sequences, API sequences, or other multimodal information that can reflect the software's intrinsic functionality to enhance the fine-grained classification model. Additionally, although the method has made progress in automatically generating software classification labels, human involvement is still required. Thus, future research could consider incorporating domain knowledge into large-scale pre-trained language models to enhance their reasoning capabilities and achieve the automatic generation of software label systems.

Author Contributions: Conceptualization, W.B.; methodology, W.B. and H.S.; validation, W.B. and F.K.; formal analysis, W.B. and Q.H.; investigation, W.B. and H.S.; resources, W.B. and Y.Z.; data curation, and H.S.; writing—original draft preparation, W.B.; writing—review and editing, W.B.; visualization, Q.H.; supervision, F.K.; project administration, W.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The dataset used in this paper consists of application software text descriptions crawled from popular international app stores, namely SourceForge, 360 Software Manager, Huawei AppGallery, and mi App Store. These app descriptions serve as the raw dataset for analysis.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Number of Apps Available in Leading App Store. Available online: http://www.gartner.com/newsroom/id/2592315. (accessed on 24 May 2023).
2. Liu, X.; Song, H.H.; Baldi, M.; Tan, P.-N. Macro-scale mobile app market analysis using customized hierarchical categorization. In Proceedings of the IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016; pp. 1–9.
3. 360 App Market. Available online: https://ext.se.360.cn/ (accessed on 26 May 2023).
4. Liu, L.; Comar, P.M.; Saha, S.; Tan, P.-N.; Nucci, A. Recursive nmf: Efficient label tree learning for large multi-class problems. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 11–15 November 2012; pp. 2148–2151.
5. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
6. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610. [CrossRef] [PubMed]
7. Grootendorst, M. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv* **2022**, arXiv:2203.05794.
8. Wang, T.; Wang, H.; Yin, G.; Ling, C.X.; Li, X.; Zou, P. Mining software profile across multiple repositories for hierarchical categorization. In Proceedings of the 2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, 22–28 September 2013; pp. 240–249.
9. Olabenjo, B. Applying naive bayes classification to google play apps categorization. *arXiv* **2016**, arXiv:1608.08574.
10. Kawaguchi, S.; Garg, P.K.; Matsushita, M.; Inoue, K. Mudablue: An automatic categorization system for open source repositories. In Proceedings of the 11th Asia-Pacific Software Engineering Conference, Busan, Republic of Korea, 30 November–3 December 2004; pp. 184–193.
11. Tian, K.; Revelle, M.; Poshyvanyk, D. Using latent dirichlet allocation for automatic categorization of software. In Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, Vancouver, BC, Canada, 16–17 May 2009; pp. 163–166.
12. Wang, T.; Yin, G.; Li, X.; Wang, H. Labeled topic detection of open source software from mining mass textual project profiles. In Proceedings of the First International Workshop on Software Mining, Beijing, China, 12–16 August 2012; pp. 17–24.
13. Wang, Z.; Li, G.; Chi, Y. Multi-classification of android applications based on convolutional neural networks. In Proceedings of the 4th International Conference on Computer Science and Application Engineering, Sanya, China, 20–22 October 2020; pp. 1–5.
14. Singla, K.; Mukherjee, N.; Bose, J. Multimodal Language Independent App Classification Using Images and Text. In *Natural Language Processing and Information Systems, Proceedings of the 23rd International Conference on Applications of Natural Langauge to Information Systems, NLDB 2018, Paris, France, 13–15 June 2018*; Silberztein, M., Atigui, F., Kornyshova, E., Metais, E., Meziane, F., Eds.; Springer: Cham, Switzerland, 2018; pp. 135–142.
15. Zhang, H.; Qin, J.; Wang, Y.; Ma, Y.; Yao, L.; Lei, J. Research on android multi-classification based on text. *J. Phys. Conf. Ser.* **2021**, *1828*, 012049. [CrossRef]
16. Zhou, C.; Sun, C.; Liu, Z.; Lau, F. A C-LSTM neural network for text classification. *arXiv* **2015**, arXiv:1511.08630.
17. Du, C.; Huang, L. Text classification research with attention-based recurrent neural networks. *Int. J. Comput. Commun. Control* **2018**, *13*, 50–61. [CrossRef]
18. Adhikari, A.; Ram, A.; Tang, R.; Lin, J. Docbert: Bert for document classification. *arXiv* **2019**, arXiv:1904.08398.
19. Sun, C.; Qiu, X.; Xu, Y.; Huang, X. How to fine-tune bert for text classification? In Proceedings of the Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, 18–20 October 2019; pp. 194–206.
20. Alhaj, F.; Al-Haj, A.; Sharieh, A.; Jabri, R. Improving Arabic cognitive distortion classification in Twitter using BERTopic. *Int. J. Adv. Comput. Sci. Appl.* **2022**, *13*, 854–860. [CrossRef]
21. Alawadh, H.M.; Alabrah, A.; Meraj, T.; Rauf, H.T. Semantic Features-Based Discourse Analysis Using Deceptive and Real Text Reviews. *Information* **2023**, *14*, 34. [CrossRef]
22. Kaur, K.; Kaur, P. Improving BERT model for requirements classification by bidirectional LSTM-CNN deep model. *Comput. Electr. Eng.* **2023**, *108*, 108699. [CrossRef]
23. Alawadh, H.M.; Alabrah, A.; Meraj, T.; Rauf, H.T. Attention-Enriched Mini-BERT Fake News Analyzer Using the Arabic Language. *Future Internet* **2023**, *15*, 44. [CrossRef]
24. Xie, Q.; Dai, Z.; Hovy, E.; Luong, T.; Le, Q. Unsupervised data augmentation for consistency training. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 6256–6268.
25. Sennrich, R.; Haddow, B.; Birch, A. Improving neural machine translation models with monolingual data. *arXiv* **2015**, arXiv:1511.06709.
26. Edunov, S.; Ott, M.; Auli, M.; Grangier, D. Understanding back-translation at scale. *arXiv* **2018**, arXiv:1808.09381.
27. Yu, A.W.; Dohan, D.; Luong, M.-T.; Zhao, R.; Chen, K.; Norouzi, M.; Le, Q.V. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv* **2018**, arXiv:1804.09541.

28. Xia, T.; Wang, Y.; Tian, Y.; Chang, Y. Using prior knowledge to guide bert's attention in semantic textual matching tasks. In Proceedings of the Web Conference 2021 (WWW'21), Ljubljana, Slovenia, 12–16 April 2021; pp. 2466–2475.
29. Reimers, N.; Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv* **2019**, arXiv:1908.10084.
30. McInnes, L.; Healy, J.; Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv* **2018**, arXiv:1802.03426.
31. McInnes, L.; Healy, J.; Astels, S. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* **2017**, *2*, 205. [CrossRef]
32. Carbonell, J.; Goldstein, J. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, VIC, Australia, 24–28 August 1998; pp. 335–336.
33. Hamid, O.H. ChatGPT and the Chinese Room Argument: An Eloquent AI Conversationalist Lacking True Understanding and Consciousness. In Proceedings of the 2023 9th International Conference on Information Technology Trends (ITT), Dubai, United Arab Emirates, 9–10 September 2023; pp. 238–241.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.