European Journal of Molecular & Clinical Medicine
ISSN 2515-8260 Volume 10, Issue 03, 2023

MODELLING TO CLASSIFY SAMPLE PROBLEM SETS FOR FIRST-YEAR COMPUTER SCIENCE COURSES

Ms D S CH S HARINI, Associate Professor, kschsharini1225@gmail.com, Rishi UBR Women's College, Kukatpally, Hyderabad 500085

Abstract-

Manually determining concepts present in a group of questions is a challenging and time-consuming process. However, the process is an essential step while modeling a virtual learning environment since a mapping between concepts and questions using mastery level assessment and recommendation engines is required. In this article, we investigated unsupervised semantic models (known as topic modeling techniques) to assist computer science teachers in this task and propose a method to transform Computer Science 1 teacher-provided code solutions into representative text documents, including the code structure information. By applying nonnegative matrix factorization and latent Dirichlet allocation techniques, we extract the underlying relationship between questions and validate the results using an external dataset. We consider the interpretability of the learned concepts using 14 university professors' data, and the results confirm six semantically coherent clusters using the current dataset. Moreover, the six topics comprise the main concepts present in the test dataset, achieving 0.75 in the normalized pointwise mutual information metric. The metric correlates with human ratings, making the proposed method useful and providing semantics for large amounts of unannotated code.

1.INTRODUCTION

Measuring Students knowledge requires an understanding of which educational concepts are needed to answer each question. Recently, open online courses and intel- ligent tutoring systems are widely adopted learning environ- ments. Their popularity increases the demand for tools to map questions to concepts correctly since students' mastery level assessment and next steps recommendation depend on these mappings.

However, manually identifying the concepts required to answer the questions can be time consuming and difficult increasing the need for tools to assist teachers in the tasks

Desmarais [1] suggested that even partial automation of the process can be highly desirable. Besides decreasing the man ual labeling required from the experts, the process automation also results in a more objective and replicable mapping Applying supervised machine learning-based solutions is no entirely appropriate because it requires a considerable amoun of labeled data. Also, a question can relate to multiple con cepts, increasing the complexity of the labeling task. The tra ditional unsupervised learning methods, such as K-means and hierarchical clustering, are also not suitable for this tas because it is hard to determine each cluster's features.

This article proposes unsupervised semantic methods known as topic modeling techniques [2]-[5], as more inter pretable methods for experts, to be applied in introductor computer science problems. Specifically, we propose model ing code snippets as text documents and use topic modelin techniques to extract and improve

the semantic relationship between them. providing technology to support concept iden tification experts.

Our key research questions are summarized as follows.

1) How can semantic relationships be extracted and struc tured from code?

2) How can humans read, interpret, and use the extracte relationships?

The main contributions of this article include the following

1) a tokenization structure to transform raw code snippet into a document-term matrix:

2) a code-clustering method to optimize positively corre lated metrics for human interpretability:

3) experts validation, illustrating how the propose method can support questions exercise labeling usin each topic's terms.

The proposed code-clustering pipeline builds a document- term matrix with a code tokenizer by comparing various meth- ods, including clustering algorithms. We compare nonnegative matrix factorization (NMF) [2], latent Dirichlet allocation (LDA) [3], and K-means [6], as a baseline. The models were evaluated with the UMass and UCI coherence metrics [7]-[10] using the top-5 and top-10 terms. According to the metrics scores, we selected the two best-ranked models using Fagin's algorithm [11]. The LDA-based clustering approach provides the most interpretable results from these models. Fourteen pro- fessors manually contextualize the LDA-based clustering in the Computer Science I (CS1) domain, demonstrating how the proposed method could be used to facilitate the clusters' interpretability.

The next section begins by reviewing the manual, super- vised, and unsupervised concept identification approaches. Code-clustering techniques inside the educational data mining (EDM) and software engineering contexts are reviewed together with existing LDA proposals to handle short texts. We describe a proposed method to cluster code in Section III. The main challenges facing the CSI context are the answers' small size and document-term matrix sparsity. Our proposed code tokenizer overcame these problems using code structure information to augment the corpus. The results are shown in Section IV, where the best two clustering schemes are ana- lyzed based on the coherence evaluation metrics. This section also demonstrates how professors can get an overview of the required concepts from these results. Finally, Section V con- cludes this article with future work directions.

The existing methods to identify concepts from a set of CS1 exercises involve manual work and input from experts [12], [13]. For example, Sheard et al. [12] characterized introduc- tory programming examination questions according to their concept areas, question style, and required skills. Participants manually classified the questions and the determined topics covered alongside the necessary skill levels to solve them. Nonetheless, applying a successful approach in a different set of exercises requires a new manual labeling stage, which may not be achievable.

One strategy to overcome this issue and minimize the domain experts' workload is to apply supervised learning. Pre- vious research in question classifications used

supervised learning to classify questions according to the level of diffi- culty [14], Bloom's taxonomy [15], answer type [16], and sub- ject [17]. In Godea et al. [16], the features are derived from the questions, using part-of-speech tags, word embeddings, inter- class correlations, and manual annotation. Supraja et al. [15] used a grid search to analyze different combinations of weight schemes and methods to find the best set of parameters to build a supervised model to classify questions given Bloom's Taxonomy. Its main cost is the manual annotation of all labels. which is impractical when applying to large datasets. Unsu- pervised learning can group similar items without a predefined label, but it is harder to ascertain the results since there is no objective goal to analyze, and evaluating the clustering out- comes becomes a subjective task. Unsupervised learning tech- niques have been used to address EDM problems [18]-[22]. For example, Trivedi et al. [23] used spectral clustering with linear regression to predict student performance. In the ques- tions' classification context, an unsupervised approach using K-means, as a clustering algorithm [24], was proposed to group similar learning objects (such as handouts, exercises, comple- mentary readings, and suggested activities). Still, K-means does not provide a list of features that best characterize each cluster, making the expert infer them manually by reading a

Although the LDA in Blei et al. [3] is a common tech- nique in topic modeling, it does not perform well in short texts (code in this context) because the traditional way of extracting terms does not provide enough textual words to characterize a specific topic [33], [34]. It is necessary to decrease the latent document-topic or word-topic spaces, making them more specific for each context. Hsiao et al. [35], [36] proposed a topic-facet LDA model using sen- tence LDA (SLDA) with a facet representing a more spe- cific topic and all words from a sentence belonging to the same facet. Zhao et al. [37] decreased the latent space by creating a common word distribution with denominated background words, which are the same for every topic. Steyers et al. [38] and Rosen-Zvi et al. [39] adopted a similar strategy. In their method, the generative process to create a document decreases the space by choosing an author and then choosing a topic. Li et al. [40] used a dis- tribution over tags to restrict the latent topic space before inferring the documents' topic distribution. Another approach to overcome the lack of textual words

which is similar to the method we adopted in this article. In our proposed tokenizer, we increase the vocabulary size (total terms) from 287 to 2388 and the average of terms per document from 23 to 137. We maintain a 95% sparsity. which agrees with the sparsity of the long-text documents from Syed and Spruit [42] and Zhao et al. [37], i.e., suc- cessfully clustered using topic modeling techniques.

III. METHODS

The methodology is categorized into the following three main tasks.

1) Generate a database by crawling different Python web tutorials.

2) Run code-clustering experiments to group exercises into topics.

3) Ask experts to contextualize the clusters into CS1 concepts.

Task 1 prepares the data to investigate the research ques- tions. Task 2 explores ways of extracting semantic relation- ships from code [research question 1 (RQI)] by proposing a code tokenizer and comparing various data transformation methods and topic modeling algorithms. In task 3, we ask pro- fessors to read and interpret task 2, giving support for RQ2. All the scripts used in this study analyses were achieved using Python and open-source Python libraries.

A. Dataset

The objective of the experiment is to find semantically related CS1 code solutions written in Python. We chose four introductory online tutorials: 1) Practice Python [43]: 2) Python School [44]: 3) Python Programming Exercises [45]: and 4) W3Resource [46] that provide both solutions and exer- cise statements. Since the sources do not have label topics or follow a course curriculum with structured syllabus topics, we work in an unsupervised environment. We crawled 54 exer- cises for the training set. The code snippets are functions with an average of nine lines/code.

For the test dataset, we collected solutions from another set of exercises given to us by the CSI professors at the Federal University of Rio de Janeiro (UFRJ). There are 65 different problems with their respective solutions in the dataset. As the training set, the code snippets are functions with an average of seven lines/codes.

B. Code-Clustering Pipeline

The code-clustering pipeline takes as input Python code snippets, which are semistructured text documents. By using topic modeling techniques, the pipeline outputs an underlying structure within the semistructured corpus. It contains the topics present in the code snippets and the most relevant words that characterize them. This article is based on the assumption that code snippets with similar CSI concepts share identical terms. Therefore, based on this assumption, the extracted topic underlying structure can be interpreted as CS1 concepts or groups of CS1 concepts present in the code snippets.

The code-clustering pipeline starts by transforming the orig- inal data to the proper format expected by the topic modeling methods. We augmented the data and constructed a matrix D (the document-term matrix) where each element D_1 , contains the weight of term w, in the document d. Then,

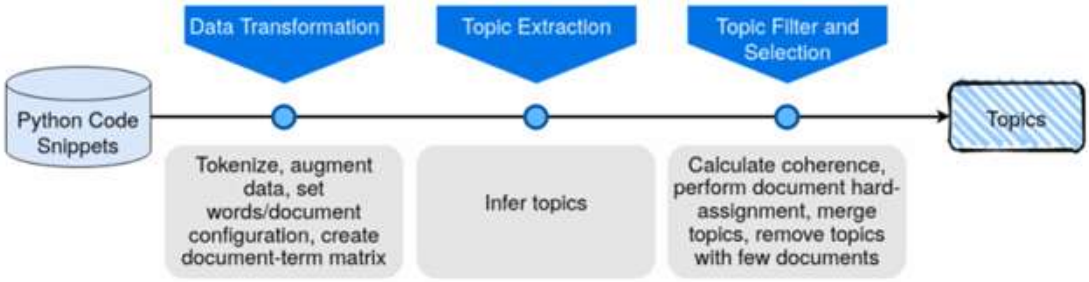

Fig. 1. Code-clustering pipeline overview [34].

using topic modeling, we calculated the relevance of each topic t for each document d, and the relevance of each term w, for each topic t. Finally, we applied a grid search and

topic coherence to choose the best models and evaluate the external corpus results. In the topic filter and selection phase, we also proc- essed the resulting topics by merging similar or removing topics with few documents. These results are presented in Section IV, while Fig. 1 illustrates the code-clustering pipe- line. External evaluation is not depicted in this overview.

1) Data Transformation: In this application, the CS1 code solutions written in Python are considered documents. The document-term matrix creation process starts by splitting each code snippet into words. The first proposed tokenizer includes only split word tokens. Henceforth, this tokenizer will be referred to as the standard tokenizer. As stated in the related work section, the LDA usually does not perform well on short texts and augmenting the corpus by adding the text's structure on semisupervised documents demonstrated improved results. We propose a new tokenizer to augment the standard tokenizer with extra features and refer to it as the augmented tokenizer. The augmented tokenizer parses the code and makes special annotations by adding extra features if the token is a number, an array (or a list), a dictionary, a string, a logical (or arithme- tic) operator, a class method, or indentation. The word itself is added to the document-term matrix if the token is a reserved word. Besides adding single tokens, this tokenizer also consid- ers bigrams and trigrams. Although the document-term matrix does not consider the terms' order, this can be enforced by adding n-grams as a matrix feature. For example, the code snippet in Fig. 2(a) is first transformed to its augmented version [see Fig. 2(b)]. Then, every single word, including the bigrams and the trigrams, are added as tokens to the document-term matrix. Table I presents some examples of the document-term matrix terms, and in total, the document is tokenized into 75 terms.

TABLE I
EXAMPLE OF DOCUMENT-TERM MATRIX

Terms	Count
is_block	3
is_indent	3
is_number	2
is_op_logic	3
if	1
is_op_logic is_number	2
is_block is_indent	3
is_op_logic is_number is_op_logic	1

A set of terms from the complete document-term matrix after augmenting and tokenizing the document from Fig. 2.

```
def light(switchA, switchB):
  if switchA == 1 and switchB == 1:
    return True
  else:
    return False
```

(a) Original code snippet

```
def light switchA switchB is_block
 is_indent if switchA is_op_logic is_number is_op_logic
             switchB is_op_logic is_number is_block
    is_indent return True
  is_dedent else is_block
    is_indent return False
  is_dedent
is_dedent
```

(b) Augmented code snippet

Fig. 2. Code snippet and its augmented version. The augmented version will be processed in the augmented tokenizer, whereas the regular one will be proc- essed in the standard tokenizer. (a) Original code snippet. (b) Augmented code snippet.

After the document-term matrix creation, we applied some transformations to enhance the document representation and decrease the matrix sparsity. First, we removed tokens with document frequency below a fixed threshold to perform fea- ture selection. This threshold was determined using a hyper-parameter grid search ranging from 5% to 50% with a 5% step. Second, we decided how to count a token frequency: either the token is counted once per document (binary

appear- ance) or every time it appears. Finally, some tokens may be more important than others. For example, the term frequency by inverse document frequency (TF-IDF) [47], [48] recalcu- lates the tokens' weights by balancing the following two factors.

1) A term that occurs in many documents should not be as important as a more exclusive term, since it does not characterize documents well.

2) A term that appears in a small number of documents may only be particular to those documents and not enough to distinguish a topic.

Yan et al. [49] proposed another way of recalculating the terms' weights. Their method (called NCut) comes from the normalized cut problem on term affinity graphs. This weight- ing scheme modifies terms' counts based on terms cooccur- rence and not on document frequency. Their experiments show NMF's performance increase on short-text clustering using the NCut weighting scheme.

2) Topic Extraction: As stated earlier, after document processing, a document-term matrix D is generated. The matrix rows represent points in an R" feature space, where n is the total number of terms, and each term corresponds to a dimension. It becomes a classical clustering problem where we expect similar documents to be in surrounding regions in space. So, clustering algorithms like K-means, hierarchical clustering, and nearest neighbors are applicable here. How- ever, for topic modeling tasks, algorithms like the NMF [2], [50] and the LDA [3] are effective since they interpret terms' counts as a set of visible variables generated from a set of hid- den variables (topics) [51], [52]. Accordingly, the documents can be modeled as a distribution of topics and topics as a dis- tribution of terms. We used the following two topic modeling techniques.

1) Nonnegative Matrix Factorization (NMF): A matrix factorization technique with a particular property of only allowing nonnegative values in its entries, which is well-suited for human interpretability [2].

2) Latent Dirichlet Allocation (LDA): A generative proba- bilistic model that describes how to create documents in a collection. Once you have a dataset, a group of already written documents, we find the distributions that create these documents. The LDA algorithm tries to backtrack this probabilistic model to find a set of topics that are likely to have generated the dataset [4]. To generate a document, we sample from two distribu- tions using the following iterative process.

a) We sample a topic for the given document (a docu- ment is a distribution of topics).

b) We sample a term from the topic sampled in step a) (a topic is a distribution of terms).

3) Topic Filter and External Evaluation: Given several document-term matrix creation options and two different topic modeling methods, we need to find the best set of hyperpara- meters. There are strategies in the literature to find a near-optimal set of models' hyperparameters, such as manual search, grid search, and random search [53]. Although random search dem- onstrates promising results in general machine learning tasks [53], Chuang et al. [54] and Wang and Blei [55] results were competitive using grid search in topic modeling tasks.

We chose to use a grid-search approach. There was a prior manual stage to define the regions in which grid search would act. Since the dataset is not large and the number of hyperparameters to try is not extensive, it is efficient to run an exhaustive search combining hyperparameters. In total, there are 1680 possible combinations: ten minimum document frequencies (ranging from 5% to 50% with 5% step increment), two binary appear- ance options, three token weights (counts) transformation possibilities (none, TF-IDF, and NCut), two clustering methods (LDA and NMF), and 14 number of clusters (i.e., 10 x 2 x 3 x 2x 14 = 1680). The grid search was set to search between 2 and 15 clusters (the upper bound is based on the number of concepts from Table II in Section III-C).

To determine whether topics are well defined, we can use topic coherence and pointwise mutual information (PMI) met- rics, which correlated well with human interpretability [8], [9], [56]. As explained in Section III-B2 (topic extraction), when using NMF or LDA, each topic is mapped to a list of top-N words that best define the topic. Topic coherence calcu- lates the ratio between the cooccurrence of these top-N words and their total occurrence. The assumption is that the words that best characterize a topic often appear together if a topic is well defined. This article applied two types of topic coherence metrics: UCI [7] and UMass [8]. The UCI metric based on PMI is calculated using an external validation source. The PMI can be substituted using normalized PMI (NPMI) to bet- ter correlate with humans' ratings [9]. The UMass metric uses the conditional probability of one word occurring given that one other high-ranked word occurred and can be measured using the modeled corpus, without depending on an external reference corpus. We used the UMass coherence to choose the best models since it is an internal validation metric (it only evaluates the clustered data). To assess the models, we used an external dataset with the UCI NPMI metric.

TABLE II
LIST OF CS1 CONCEPTS

1. Syntax	6. Logic	11. Conditional
2. Assignment	7. Data type: string	12. Loop
3. Data type: number	8. Data type: array	13. Nested loop
4. Data type: boolean	9. Data type: tuple	14. Function
5. Math	10. Data type: dict	15. Recursion

Defining P(w) as the probability of the term w, occurring and P(wi, w,) as the probability of terms w, and w, cooccur- ring, we calculated the coherence for a single topic t using (1)-(3). In this article, the topic coherence for a single topic was calculated using top-5 and top-10 terms. After calculating each topic's coherence in a single hyperparameter combina- tion, this combination's coherence was reported as the median of all topic coherence

$$C_{UMass}(W) = \sum_{i=1}^{N} \sum_{j=1}^{N} \log \frac{P(w_i, w_j) + \epsilon}{P(w_i)} \quad (1)$$

$$C_{UCI}(W) = \sum_{i=1}^{N} \sum_{j=1}^{N} \mathrm{NPMI}(w_i, w_j) \quad (2)$$

$$\mathrm{NPMI}(W) = \frac{\log \frac{P(w_i, w_j)+\epsilon}{P(w_i)P(w_j)}}{-\log(P(w_i, w_j) + \epsilon)} \quad (3)$$

where W = (w, ...) are the top-N terms for calculat- ing the coherence. An value of 0.01 was used to avoid taking a zero logarithm.

We performed hard assignment to cluster documents by topic by assigning each document to the topic with the most

relevance (weight) in the document-topic matrix. The hard assignment was achieved with minimal loss of information when a topic strongly characterized a document. In addition to assigning documents to topic clusters, the set of features/terms that best characterize each cluster/topic were extracted for fur- ther analysis.

C. Topics Contextualization

To relate concepts and topics, we first defined the most commonly seen concepts in CS1 exercises. The following four references were used to create a list of concepts com- monly used in CS1 courses.

1) Computer Science Curricula 2013 [57]: A document jointly built by the Association of Computer Machinery and the IEEE Computer Society. The document recom- mends curricular guidelines for computer science edu- cation, which we used as the main concept list. We used the papers in items 2)-4) to improve it.

2) Exploring Programming Assessment Instruments: A classification scheme for examination questions [12]: creates a classification scheme characterizing exam. questions by their concept areas, question style, and skills a student needs to solve them. We used the list of the proposed concepts as a second source to enhance the main list.

Then, to interpret the meaning of the topics, we asked 14 pro- fessors to perform three tasks. The professors (with 2-20 years of teaching experience) teach CSI or other programming- related subjects.

1) Theme Identification: We present some code snippets belonging to the topic and found essential tokens for each topic. The professors were asked to label each topic with free-text descriptions. We tokenized the descriptions and counted the terms. We also created the topic titles based on the terms that appeared more fre- quently in the descriptions.

TABLE III
SET OF HYPERPARAMETERS FOR EACH EXPERIMENT

	Min DF	Binary	Vectorizer	Method	# of clusters
Exp. 1	0.35	True	NCut	NMF	7
Exp. 2	0.05	True	Count	LDA	12

TABLE IV
NUMBER OF DOCUMENTS PER TOPIC IN EXPERIMENT 1

Topics	1	2	3	4	5	6	7
# of documents	5	4	26	19	0	0	0

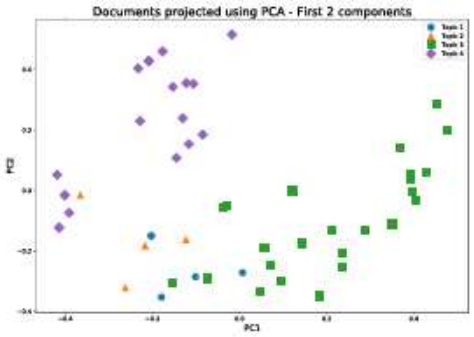

Fig. 3. Documents projected into the first two dimensions using PCA. Points with the same color and marker belong to the same cluster.

IV. RESULTS AND DISCUSSION

We run each hyperparameter combination from the 1680 possibilities ten times and calculated their average coherence and standard deviation. Next, the two best-ranked results are analyzed. They were calculated using Fagin's algorithm [11] for top-5 and top-10 terms UMass coherence. Table III shows the set of hyperparameters for each experiment.

A. Experiment 1

After the document-term matrix factorization, we hard assigned each document to the topic with the highest rele- vance (highest weight in the document-topic distribution). Table IV shows the number of

documents assigned per topic. After assigning each document to its related topic in this experiment, the documents are only assigned to four of the seven topics. Fig. 3 shows the documents projected to two dimensions using principal component analysis (PCA).

Using a minimum document frequency of 35% kept only 23 valid terms. Fig. 4 shows the essential terms per topic where the terms that are exclusively important for a single topic (a term is vital if it is above the 75th percentile of all weights) are denoted in green. In this plot, topics 3 and 4 share almost all terms. By adjusting the document-term matrix values using the NCut vectorizer, the factorization split topics 3 and 4 using the conditional if term. Topic 4 is exclusive for code snippets that are solved using conditional statements, whereas topic 3 comprises the opposite.

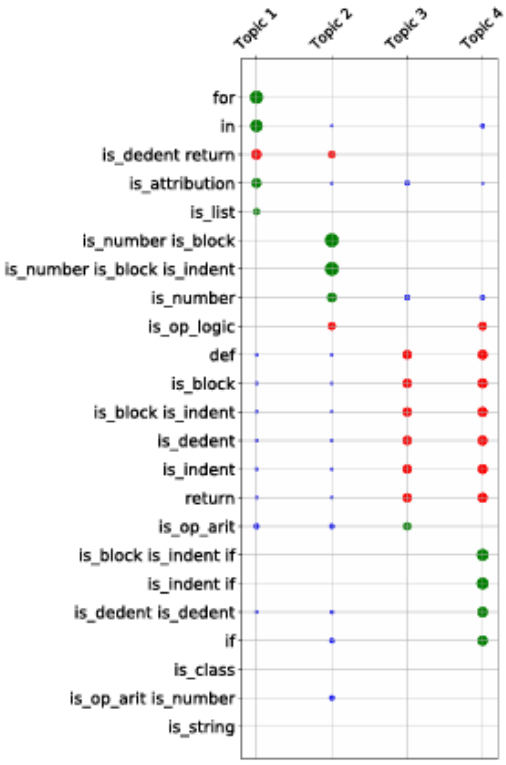

Fig. 4. Term importance for the four most populated topics. Each row represents a term, the size of the point corresponds to the term's weight for the topic, and red points are the points above the 75th percentile of all weights. The green points denote words above the 75th percentile limit on only one of the four topics.

Fig. 5 shows the topic distribution per document. As explained in Section III-B2 (topic extraction), distribution over topics describes a document. Darker cells imply that the topic characterizes a document better. As stated before, if a topic strongly characterizes a document, then we can hard assign it to a single topic. However, Fig. 5 shows that most documents assigned to topics 1 and 2 (top part of the plot) spread throughout the topics. It suggests we have to combine the most important terms for each topic to interpret these code snippets.

Analyzing the code snippets from topic 1, they combine for- loops with conditional statements. Topic 2 is a mixture con- taining the code snippets that do not belong to any other topic.

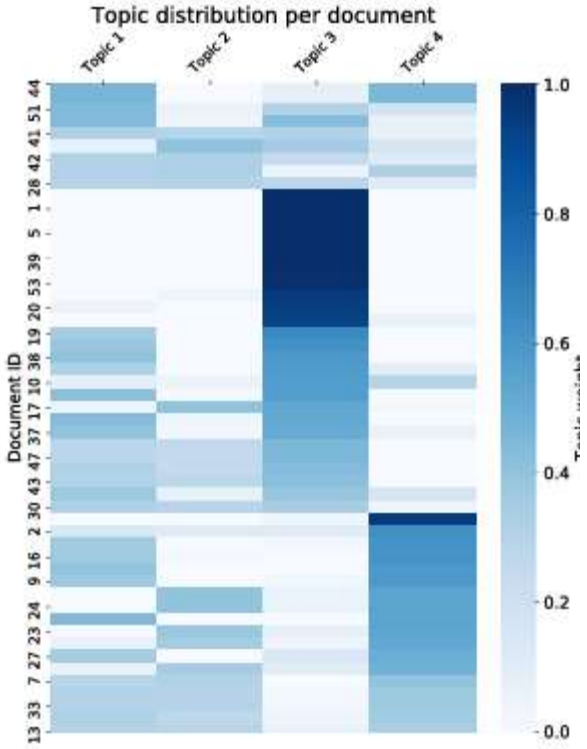

Fig. 5. Topic distribution per document. Darker cells indicate a description of the document by the topic.

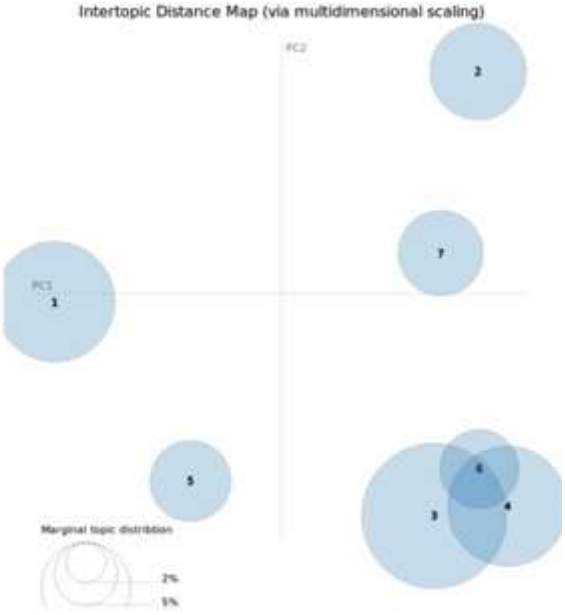

Fig. 6. Topics are represented as circles proportional to the number o whose weights are most associated with the topic and the distance betw circles is the intertopic distance.

TABLE V
NUMBER OF DOCUMENTS PER TOPIC IN EXPERIMENT 2

Topics	1	2	3	4	5	6	7	8	9	10	11	12
# of documents	2	1	1	2	1	13	1	14	0	7	0	12

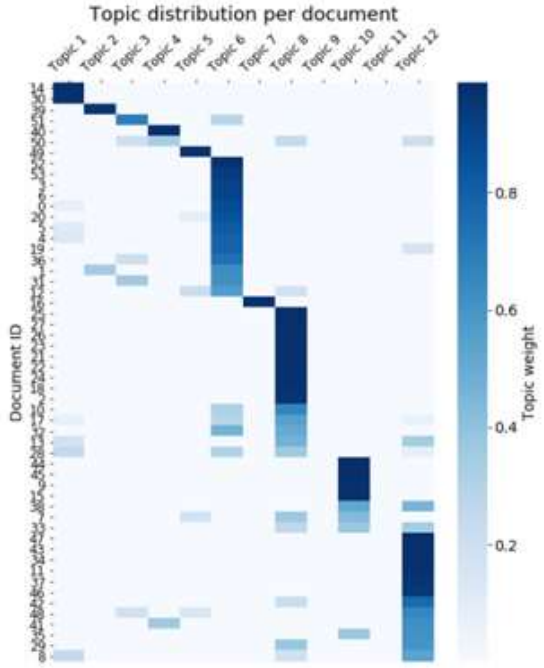

Fig. 7. Topic distribution per document. Darker cells indicate a better description of the document by the topic.

Fig. 6, using the LDAVis tool [65], calculates the topics' distance and projects them to 2-D using principal coordinate analysis. Topics 3 and 4 are located close to each other, and they correspond to 45% of the terms and 83% of the docu- ments. Fig. 6 also validates that these topics are not that differ- ent when their crucial terms are analyzed. Still, the conditional statements that characterize topic 4 are enough to produce a linearly separable 2-D data projection, except for a few out- liers, as shown in Fig. 3.

B. Experiment 2

Table V shows the number of assigned documents per topic with hyperparameters combination producing a more uniform grouping scheme than the previous one.

Although we initially set 12 clusters, two of them (topics 9 and 11) are empty after assigning each document to the topic with the highest relevance (weight). Topics 6, 8, 10, and 12 have the largest number of documents. Fig. 7 shows the topic per document distribution where the topics better characterize each document.

Fig. 8 shows the intertopic map. The main topics 6, 8, 10. and 12 correspond to 85% of the documents and 77.4% of the terms and we do not observe any main topic overlap in this plot. The next subsections analyze these main topics in detail. Topics 2 and 4 will also be analyzed since they occupy a different space on the map. This step belongs to the topic filter and selection phase from the code-clustering pipeline depicted in Fig. 1. After hard assigning the documents to the clusters (removing topics 9 and 11), merging topics 2 and 4, and removing topics with a few documents (less than three docu- ments per topic: topics 1, 3, 5, and 7), it resulted in five con- ceptual clusters (six from the original topics in total) to be analyzed in detail.

1) Topic 8 is strongly characterized by conditional state- ments, logical operators, and Boolean values.

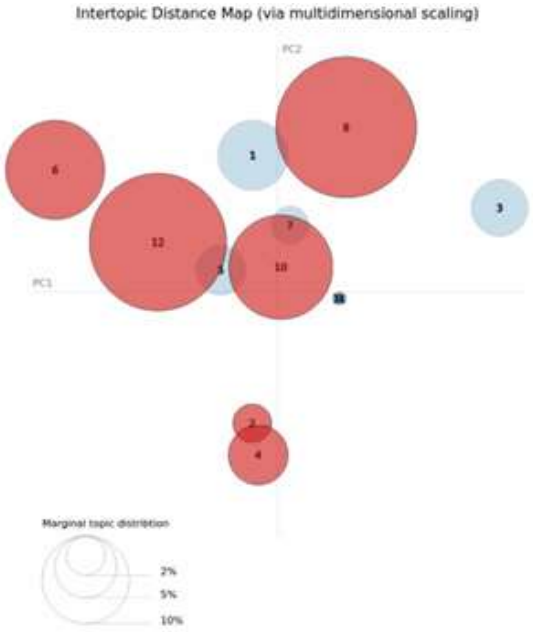

Fig. 8. Intertopic distance map for experiment 2. Topics are represented as circles proportional to the number of terms whose weights are most associated with the topic and the distance between the circles is the intertopic distance. Topics in red are further discussed in detail.

TABLE VI
FIVE MOST RELEVANT TERMS FOR EACH OF THE ANALYZED TOPICS

Topics 2 & 4	Topic 6	Topic 8	Topic 10	Topic 12
split	is_op_arit	is_op_logic	append	range
is_string	is_indent	if	is_list	is_op_arit
join	def	else	for	for
for	is_number	True	is_attribution	is_number
if	len	False	if	is_attribution

The terms starting with "is" are the special annotated terms explained in Section III-B1 (Data transformation).

```
def sort_csv(csv):
    items = csv.split(', ')
    items.sort()
    return ", ".join(items)
```

(a) Topic 2

```
def sort_dedupe(words):
    items = words.split(' ')
    items_dedupe = []
    for word in items:
        if word not in items_dedupe:
            items_dedupe.append(word)
    items_dedupe.sort()
    return " ".join(items_dedupe)
```

(b) Topic 4

```
import math
def formula(D):
    C = 50
    H = 30
    Q = round(math.sqrt(2*C*D/float(H)))
    return Q
```

(c) Topic 6

```
def is_prime(number):
    '''Returns True for prime numbers, False otherwise'''
    #Edge Cases
    if number == 1:
        prime = False
    elif number == 2:
        prime = True
    #All other primes
    else:
        prime = True
        for check_number in range(2, int(number/2)+1):
            if number % check_number == 0:
                prime = False
                break
        return prime
```

(d) Topic 8

```
def dedupe(dup_list):
    nodup_list = []
    for i in dup_list:
        if i not in nodup_list:
            nodup_list.append(i)
    return nodup_list
```

(e) Topic 10

```
def fatorial(number):
    total = 1

    for i in range(number, 1, -1):
        total = total * i

    return total
```

(f) Topic 12

Fig. 9. Example of code snippets belonging to each topic. In red are the relevant terms for each topic.

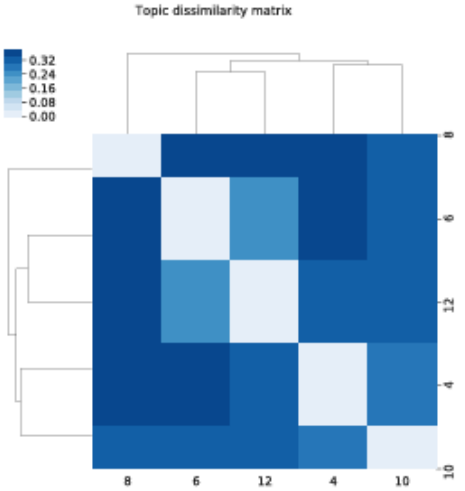

Fig. 10. Topic dissimilarity matrix. The darker the cell the more distant the topics are from each other.

TABLE VII
NUMBER OF TEST SET DOCUMENTS PER TOPIC FOR EXPERIMENT 2

Topics	1	2	3	4	5	6	7	8	9	10	11	12
# of documents	2	4	0	1	0	23	0	17	0	3	0	15

TABLE VIII
MEAN AND STANDARD DEVIATION OF UCI COHERENCE USING NPMI

	C_{UCI} (NPMI)	
	Top-5 terms	Top-10 terms
NMF (Experiment 1 with augmented tokenizer)	0.73 (0.03)	**0.83 (0.02)**
LDA (Experiment 2 with augmented tokenizer)	**0.75 (0.06)**	0.76 (0.04)
K-Means (with augmented tokenizer)	0.55 (0.09)	0.53 (0.03)
LDA (best result for standard tokenizer)	0.53 (0.03)	—

Values in bold represent the best result for each metric.

(including topics 2 and 4) valid topics to understand if the topics are representative of the possible concepts present in an unseen code. We assigned each code to the topic with the highest weight as we did for the training set. Table VII shows the number of assigned documents per topic. Except for two documents, all the others belong to one of the six valid topics. It confirms that the different topics (the ones considered invalid) detect specific code traits and not their general con- cepts. It is important to notice that topic modeling is a soft clustering

technique: a document has a probability of belong- ing to each topic and can be associated with more than one. So, a document can be related to the main topic with its speci- ficity related to minor ones.

C. Coherence Evaluation

Both experiments were analyzed using the UCI coherence metric with NPMI [9], as described in Section III-B3 (topic filter and external evaluation), to validate how well the proposed meth- odology performs in an external dataset. Although AST trees have been used to cluster code, they do not provide an intuitive way to analyze the important features besides reading it. We compare our results with a K-means clustering method using the proposed augmented tokenizer and logistic regression to extract the important features per cluster as a baseline. We also com- pared our best results using the standard tokenizer instead of our proposed tokenizer. We used k= 5 for K-means since there were five main conceptual clusters found in the LDA. We ran each method 100 times and averaged their UCI coherence metric. Statistical difference was measured using the Mann-Whitney U test [66], and all the results were statistically significant with $p < 0.001$. Table VIII reports the mean and standard deviation for each experiment. In the UCI coherence with NPMI metric. the values are bounded between 1 and -1, where 1 means that the top words only occur together, 0 means that they are distributed as expected under independence, and -1 means that they only occur separately. The UCI coherence for the standard tokenizer using the top-10 terms could not be measured because there were no important top-10 term pairwise combinations in this setting that appeared in at least one document. The NMF experiment with the augmented tokenizer considering the top-10 terms dem- onstrated the best UCI occurrence metric, followed by the LDA experiment, which had the best performance considering the top-5 terms.

D. Discussion About Experiments 1 and 2

TABLE IX
RELATION BETWEEN THE FOUND TOPICS AND THE MAIN CS1 CONCEPT RELATED TO THE TOPIC

Topic	Main Concept	Agreement inside cluster
2 & 4 - String manipulation	7. Data type: string	66.7%
6 - Math functions	14. Function	50%
8 - Conditional structure	11. Conditional	86.7%
10 - List loops	12. Loop	71.4%
12 - Math loops	12. Loop	75%

We found both experiments to have their main concepts in a few clusters (two main clusters in Experiment 1 and six main clusters in Experiment 2). The remaining clusters are associ- ated with code specificity. In the case of Experiment 1, using NCut and a high document frequency threshold, the topic modeling from Experiment 1 focused on finding structures with high volume and cooccurrence rates, resulting in separa- tion of the if/else structure from the rest. The conditional struc- ture was first separated from a hierarchical perspective, and the remaining structures were all grouped in a cluster. In Experiment 2, the conditional topic (topic 8) is also the fur- thest from the other topics. As shown in the hierarchical clus- tering of Fig. 9, this topic is the last one to be aggregated (or the first one to be separated). The common code snippets between the conditional clusters in each experiment also vali- date this result. From the 14 code snippets associated with the conditional topic (topic 8) in Experiment 2, 11 of them

(79%) belong to the conditional topic in Experiment 1 (topic 4). Therefore, Experiment 2 demonstrates more granularity than Experiment L.

E. Topics Contextualization

1) Concept Identification: Each professor was asked to associate up to three concepts (from the 15 available in Table II) to each presented code. Four professors ana lyzed each code. In 37 of the 54 code snippets, there was at least one concept in common between all four professors. In 53 of the 54 code snippets, at least one concept was common between three out of the four pro- fessors (75%). Therefore, we decided to use the 75% threshold of the agreement to relate the exercises' con- cepts. The concepts in each topic were aggregated to provide an overview of the main concepts needed to solve the cluster's problems, as summarized in Table IX.

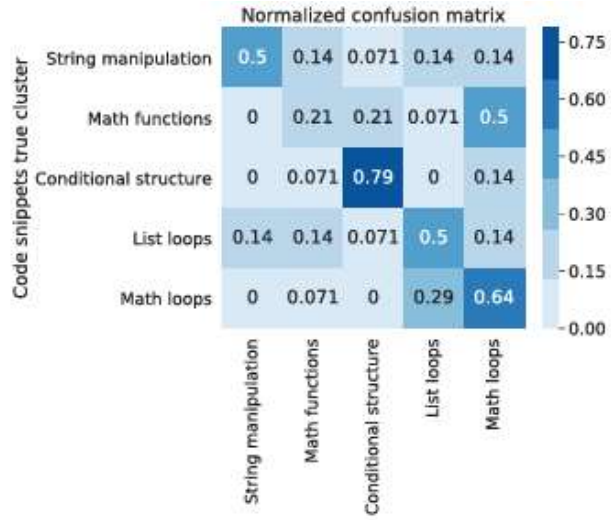

Fig. 11. Normalized confusion matrix for the intruder-identification task.

2) Intruder Identification: Four code snippets were pre-

a) The "conditional structure" topic performed well, with the intruder code being identified 79% of the time, meaning that identifying a code snippet from a different cluster can be done 4 out of 5 times.

b) The intruder code inside the "math loops topic was identified 2 out of 3 times (64%), being confused with "list loops" the last third of the time. These topics work on the same main concept, as seen in the concept identification task.

V. CONCLUSION

Based on the evaluation metric, our proposed method found semantically related code-clustering schemes suitable for human interpretability with minimal supervision, giving sup port for RQI. The method is expected to provide semantics for large amounts of unannotated code.

Although code clustering in the CSI context has been widely applied using the AST trees, the advantages of working with topic modeling are the terms per topic results that may help experts better assess each cluster's contents. The method- ology has also been shown to overcome the small-sized code snippets challenge by extending the tokenizer to augment the corpus with the code structure. The standard tokenizer could not create semantically related topics, but adding structural information, as features: indents and data types, and enforcing the order using n-grams enriched the code representation and found topics suitable for human interpretability. For example, augmenting the corpus with structural information as indents/ blocks (in Python, indents indicate how deep a block of code is; other languages like C++ and Java could count the number of "" and "") help to separate single loops from nested loops. Combining trigrams (to enforce order) with structural information can distinguish subtle

differences in precondition and postcondition loops. Notice that postcondition loops do not exist in Python, so we could not verify this specific assumption. In our dataset, we expect trigram tokens to be enough to capture these varieties because a typical CS1 solu- tion does not have more than three or four nested structures. Still, it may limit our model in identifying large nested struc- tures on more complex code. Also, even though there is a recursion concept in the concepts list, there was no exercise using this technique in our dataset to verify how it would be clustered.

ACKNOWLEDGMENT

The authors would like to thank all the professors who contributed to the research by providing CS1 problems and respective solu- tions or evaluating the results.

REFERENCES

[1] M. C. Desmarais, "Mapping question items to skills with non-negative matrix factorization," SIGKDD Explorations Newslett., vol. 13, no. 2, pp. 30-36. May 2012, doi: 10.1145/2207243.2207248 [2] D. D. Lee and H. S. Seung. "Learning the parts of objects by non-negative matrix factorization." Nature, vol. 401, pp. 788-791. Oct 1999. doi: 10.1038/44565, [3] D. M. Blei. A. Y. Ng, and M. 1. Jordan. "Latent Dirichlet allocation," J Mach Learn. Res., vol. 3, pp. 993-1022. Jun. 2003.

[4] M. Steyvers and T. Griffiths, "Probabilistic topic models." in Handbook of Latent Semantic Analysis. 1st ed., T. Landauer, S. D. McNamara, and W. Kintsch, Eds. New York, NY, USA Psychology Press, 2007, ch. 21. pp. 427-448, doi: 10-4324/9780203936309

[5] T. Hofmann. "Probabilistic latent semantic analysis" in Proc. 15th Conf Uncertainty Arti. Intell., 1999, pp. 280-206.

[6] S. Lloyd. Least squares quantization in PCM. IEEE Trans. lot. The- ory, vol. 28, no. 2, pp. 129-137. Mar. 1982.

[7] D. Newman, J. H. Lau, K. Grieser, and T. Baldwin, Automatic evalua tion of topic coherence in Proc. Ann Conf North Ainer. Chapter