Exploring Programming Semantic Analytics with Deep Learning Models

Yihan Lu
School of Computing, Informatics & Decision Systems Engineering, Arizona State University
Tempe, Arizona
lyihan@asu.edu

I-Han Hsiao
School of Computing, Informatics & Decision Systems Engineering, Arizona State University
Tempe, Arizona
Sharon.Hsiao@asu.edu

ABSTRACT

There are numerous studies have reported the effectiveness of example-based programming learning. However, less is explored recommending code examples with advanced Machine Learning-based models. In this work, we propose a new method to explore the semantic analytics between programming codes and the annotations. We hypothesize that these semantics analytics will capture mass amount of valuable information that can be used as features to build predictive models. We evaluated the proposed semantic analytics extraction method with multiple deep learning algorithms. Results showed that deep learning models outperformed other models and baseline in most cases. Further analysis indicated that in special cases, the proposed method outperformed deep learning models by restricting false-positive classifications.

KEYWORDS

Coding concept detection, Programming semantics, Text based classification, Semantic modeling, deep learning

ACM Reference Format:
Yihan Lu and I-Han Hsiao. 2019. Exploring Programming Semantic Analytics with Deep Learning Models. In *The 9th International Learning Analytics & Knowledge Conference (LAK19), March 4–8, 2019, Tempe, AZ, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3303772.3303823

1 INTRODUCTION

The demand of programming in industry is becoming inevitable. However, learning programming involves a variety of complex cognitive activities, from conceptual knowledge construction to basic structural operations, program design, programming understanding, modifying, debugging, and documenting [15, 17]. It is especially challenging for novices [12]. Novices face to a serial of difficulties while learning programming, including the misunderstanding of concepts, the large time cost in learning, the lack of efficient tutoring, etc[11].

While machine learning techniques have shown that students can benefit from different ways including intelligent tutoring[1, 3], virtual classroom[5], searching assistance[13], there are still elements overlooked or not used in applications in programming education, one of which is the *programming code annotations*. It is a common practice for programmers to leave annotations during program development. The annotation content can range from code descriptions, uses and unused of codes, expected/unexpected outcomes, copyrights, algorithmic logics, limitations, comments on the implementation. Most of the annotated documentations are predominantly being used as the archive of the coding events for limited developers. We hypothesize that these annotations captured mass amount of valuable information based on the types, locations, purposes, and understandings. This information can be further used to reduce the cognitive load[18] for coders. Therefore, by extracting and modeling the semantics of the annotations could enable multiple applications in the educational context, for instance, further estimation of the coders' intention, examination of the problem-solving approaches, example code recommendations, etc[2].

Due to the complexity of annotations may consist of compound information (i.e. coder's interpretation), it is challenging to accurately extract the connection between annotations and codes. Thus, it increases the difficulty to model the semantics in order to further reference code to comments. In this work, we propose to use concept as a connection between annotation and code, which can be extracted from code with parser. Then we conducted experiment to test multiple models identifying concepts from annotation semantics. The results indicated neural network outperformed other models in most cases; LightGBM was more efficient in time by trading off accuracy.

The contribution of this research is that we built up a serial of methods that extract semantics from programming codes and comments, and utilized deep learning models to predict concepts involved in corresponding comments of codes. These predictive models enable several educational application opportunities, such as personalized recommenders for coding support or intelligent tutors. Essentially, utilizing the semantic analytics to supply adaptive instructional material, code examples, or machine-generated natural language content that is appropriately for the learner at the time. Furthermore, we evaluated the accuracy performance for different models, and identified the best situations for each model by analyzing cases in the experiment.

2 LITERATURE REVIEW

There have also been a few studies investigated the relations between natural languages semantics and programming languages. A parser-based code example recommender was developed by

Singh[19] for educational purpose. In this recommender, a user needed to provide a piece of code, then parser was used to analyze the structure of programming code and extract features from the code including concepts used, length, methods used, etc. These features were used to identify similar code recommended to students for reference. This work is closely related to ours, we reused Singh's concept detection mechanism, but further utilized comment from user, instead of treating a piece of code as query.

Tan et al.[20] developed "aComment" to detect bugs in program with comments and code. They utilized keyword detection in comment to identify specific bug related sentences and determine whether it is a potential bug in the corresponding code. Then the code was scanned in the function call level with a set of principles to detect real violations. Our work also considered connecting comments and codes, but the method we used is more adaptive, and the purpose was in education.

Corazza[4] researched the coherence between comments and the corresponding codes. In this work the comments were analyzed with natural language processing tools and compared to the corresponding function names to examine the coherence lexical similarity between comment and code. Corazza's work proved that the code and comment coherence is higher when they are lexical similar, which provided us the precondition to utilize the connection between comment and code.

Hsiao et al.[7] evaluated a variation of topic model by integrating NLP-based short texts and programming language syntactical features. Such method has been applied to extract semantics from programming exams[8] and programming discussion forums[6].

Cory et al.[3] proposed an intelligent tutoring system based on internet for programming learner. This tutoring system utilized Bayesian network and programming problem modeling to implement personalized learning service. In this work, the problem modeling method is close to code concept extraction. However, instead of manual concept definition, we used automatic extraction.

Haixun et al. attempted to extract semantic knowledge from short text[9]. From the study, they also indicated the difficulties in understanding short text, including the lack of syntax elements, the shortage in state-of-the-art tagging, and the impact of noise.

Another study conducted by Howard aimed to identify semantically similar words from comment-code mappings[16], which utilized a mapping method close to ours, but focused on the connection between annotations and function signatures. In our work, we designed a programming language parser that analyzes code and annotation connections and extracts concepts from the context. The capability to preserve the code and annotation connections permits to capture the semantic meaning of the code in a finer-grained level, because the scale of code analysis is ranged from the whole code file to each single word.

Among the related works, our study is unique since we focused on the educational value of the connection between comments and code. To extract the value from comment text, we utilized a serial of nature language processing methods; To extract the information from code, we applied parsing techniques on programming code.

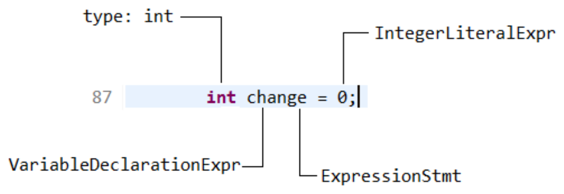

```
87    int change = 0;
```

Figure 1: Example of concept extraction from a code line.

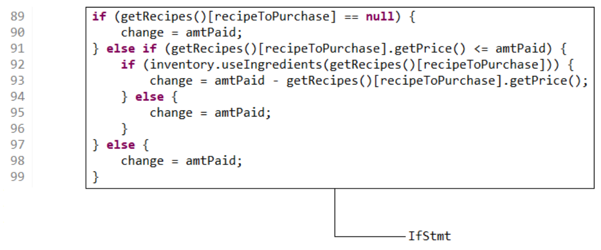

```
89  if (getRecipes()[recipeToPurchase] == null) {
90      change = amtPaid;
91  } else if (getRecipes()[recipeToPurchase].getPrice() <= amtPaid) {
92      if (inventory.useIngredients(getRecipes()[recipeToPurchase])) {
93          change = amtPaid - getRecipes()[recipeToPurchase].getPrice();
94      } else {
95          change = amtPaid;
96      }
97  } else {
98      change = amtPaid;
99  }
```

Figure 2: Example of Concept belongs to a block of code.

3 METHODOLOGY

3.1 Code Concept Extraction

The first step of this work was to extract code concepts from code blocks. The methodology of concept extraction utilized programming semantic parser. This technique recursively scanned programming code, identified structure, and detected concepts in code line or block. Fig. 2 is an example of concept extraction in a single line.

Besides the names of concept, this method also recorded the line range of each occurrence. In the last example, all four concepts belong to the line 87. A concept could also belong to a range of lines. In Figure 2, the concept "IfStmt" belongs to a block of code from line 89 to 99.

After concept extraction, each code file was turned into a set of concepts with line ranges in the file.

3.2 Code & Annotation Engineering

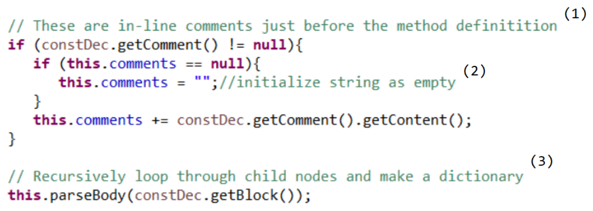
To capture code and annotation connection semantics, the adjacency relation was extracted from the line range of concepts and annotations.

Annotation text was extracted from code file with line range, and each annotation text was matched to a set of concepts according to adjacency relations. As shown in Figure 3, there are three types of line range adjacency.

3.3 Concept Prediction from Annotation

After the comment concept engineering, each annotation was mapped with a piece of code, which include a set of concepts detected. The next step was to utilize machine learning techniques to capture the connection between annotation semantics and concepts.

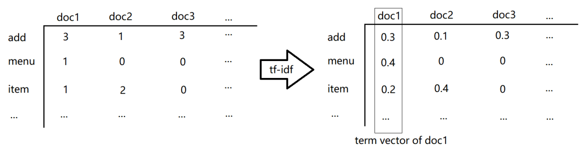
In this work, a new concept predictors was evaluated with two existing modeling algorithms in experiment. The models were also compared with random guess as baseline.



```
// These are in-line comments just before the method definition   (1)
if (constDec.getComment() != null){
   if (this.comments == null){
      this.comments = "";//initialize string as empty   (2)
   }
   this.comments += constDec.getComment().getContent();
}

// Recursively loop through child nodes and make a dictionary   (3)
this.parseBody(constDec.getBlock());
```

Figure 3: Three types of adjacency: (1) comments before a block of code statement; (2) comments follow the termination of the code in the same line; (3) comments before a single line of code.



	doc1	doc2	doc3	...
add	3	1	3	...
menu	1	0	0	...
item	1	2	0	...
...	...	...	...	...

	doc1	doc2	doc3	...
add	0.3	0.1	0.3	...
menu	0.4	0	0	...
item	0.2	0.4	0	...
...	...	...	...	...

Figure 4: An example of applying tf-idf on document-term matrix; The term vector of doc1 is highlighted.

3.3.1 *Linear Predictor.*

The linear predictor was based on an existing work[14]. In this predictor, each comment text was processed as a natural language document to extract linguistic features. After punctuation removal, stop-words removal, and stemming, we transformed the documents into a document-term matrix, and applied tf-idf algorithm on the matrix to determine the representativeness for each term in each document. For example, in Figure 4 the content of *doc1* is:

"Add a separator to the supplied menu. The separator will only be added, if additional items are added using addAction."

The term vector of doc1 is initiated as Figure 4 on the left. After applying tf-idf, the values in term vector of doc1 turn into fractions as Figure 4 on the right. Since the term "menu" does not appear in other documents, its representativeness is higher than the other terms for doc1.

By constructing linear model between each term and each concept, the connection was captured from the term-concept matrix built up from tf-idf matrix and concept-document matrix.

Further more, given a natural language comment, it was easy for the linear model to calculate the probabilities that the comment involves each concept. In this way, a high frequency concept was more likely to be determined as involve.

3.3.2 *Light gradient boosting predictor.*

Light gradient boosting predictor is based on the LightGBM decision tree proposed by Ke, etc[10]. LightGBM is a Gradient boosting decision tree (GBDT) model improved with Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). This method is proved to be more efficient while achieving almost the same accuracy compare to GBDT.

To conduct ligthGBM in annotation concept prediction, each concept is trained separately and independently. For each concept, the TF-IDF matrix is taken as feature space to capture the semantic

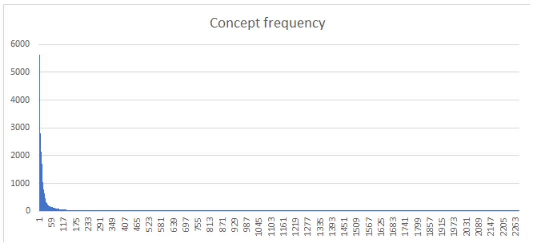


Figure 5: The frequency of concepts in dataset obeys Zipf's law.

from documents and predict whether the concept is involved. In this feature space, each column is a single feature.

In LightGBM, the importance of each feature is determined. This fact means after training, the role of each term in predicting the concept is clear in the model, the highly related term is determined. In this experiment, the parameter tuning result showed that 100 trees with 31 leaves in each had the best performance.

3.3.3 *Neural network predictor.*

Neural network is a widely used technique in image classification and natural language processing. In this work a neural network model is conducted to capture the connection between natural language semantics and code concepts.

A neural network of two layers was conducted in this work. According to the result of parameter tuning, to achieve the best performance the first layer had 100 units and the second had 20. Like LightGBM, the TF-IDF matrix of documents was taken as feature space to predict whether the concept is involved in each document.

One of the shortages in the neural network predictor is that the importance of each term is not visible. It means for each concept, it is not clear which terms are the factors indicating the concepts.

4 EVALUATION

4.1 Dataset

In this experiment, 6646 code pieces were collected from programming textbooks as the dataset, each piece of code had a corresponding comment. In this dataset, 2281 concepts were involved in total. The frequency of each concept had an exponential distribution(Figure 5), which obeys Zipf's law.

To evaluate the models, an experiment was conducted to compare the performance among neural network(NN), linear predictor, lightGBM, and random guess as baseline. In the random guess model, each concept was randomly determined as included or not in each document, the higher frequent a concept was, the higher chance it was determined as included in a document.

In the training phase, 60% of the code pieces was used as training data, and the rest 40% was used for testing. Among the 60% training data, 10% of it was used for validation. All the models used the same split of data for training and testing.

During the evaluation of all models, each concept had an independent model trained and evaluated.

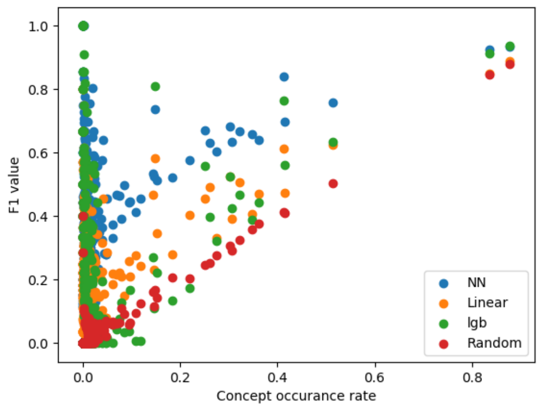


Figure 6: Concept quality comparison of NN, linear predictor, LightGBM, and random guess as baseline.

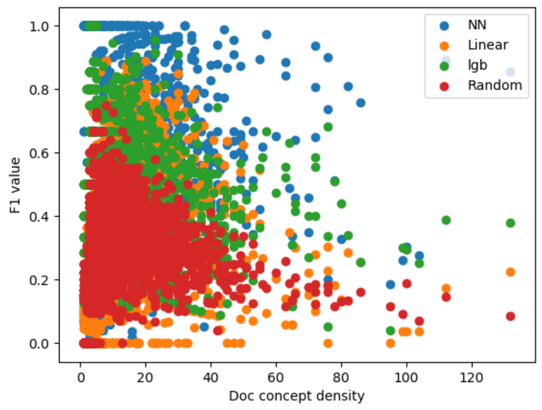


Figure 7: Document quality comparison of NN, linear predictor, LightGBM, and random guess as baseline.

4.2 Concept prediction quality

The first criteria to evaluate concept predicting models was to see for each concept, whether a model can identify all the documents including this concept. In this way, each concept can be evaluated independently for each model by f1 value. The experiment result was shown in Figure 6.

As shown in Figure 6, the neural network predictor (NN) outperformed the other models in most cases, especially for the high frequency concepts. Linear predictor performed better than lightGBM for lower frequency concepts but outperformed for higher frequency concepts. All models performed better than the random baseline.

4.3 Document prediction quality

The other criteria to evaluate models was to see whether they can identify all concepts correctly for each annotation document. The evaluation result was shown in Figure 7.

According to the result, NN was still the best model among the 4, especially for the large documents containing large number of concepts. The lightGBM outperformed linear predictor when a document included more than 50 concepts. All models outperformed the random baseline.

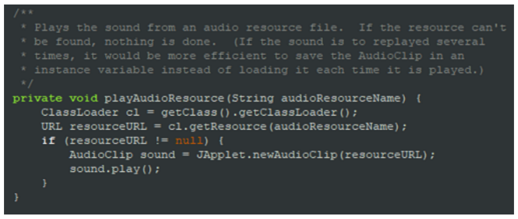
4.4 Case analysis

In this section, specific cases were identified in the dataset to analyze the reasons why each model would have false positive or true negative errors and compare the models to determine the best scenarios for each model.

4.4.1 NN perform best in deep logics.

For most concepts, the NN outperformed the other models. Concept "IfStmt" was one of these concepts, we looked into one case to uncover the reason. In Figure 8, the code and comment is shown.

In this case, the concept "IfStmt" was detected in the code part, because there was an if statement in this function implementation. In the comment describing this code, the functionality was explained for the function, including the condition of the if statement.



```
/**
 * Plays the sound from an audio resource file.  If the resource can't
 * be found, nothing is done.  (If the sound is to replayed several
 * times, it would be more efficient to save the AudioClip in an
 * instance variable instead of loading it each time it is played.)
 */
private void playAudioResource(String audioResourceName) {
    ClassLoader cl = getClass().getClassLoader();
    URL resourceURL = cl.getResource(audioResourceName);
    if (resourceURL != null) {
        AudioClip sound = JApplet.newAudioClip(resourceURL);
        sound.play();
    }
}
```

Figure 8: Example of NN outperforms.

However, only neural network successfully detected the concept "IfStmt". By analyzing each model, the result identified that the term "if" has appeared in many other comments, which were not involved with "IfStmt", so it was not a strong indicator of the concept. However, by capturing more complex combinations in text, such as the term combination of "If ... it would be ...", neural network took the advantage of deep logic, and outperformed.

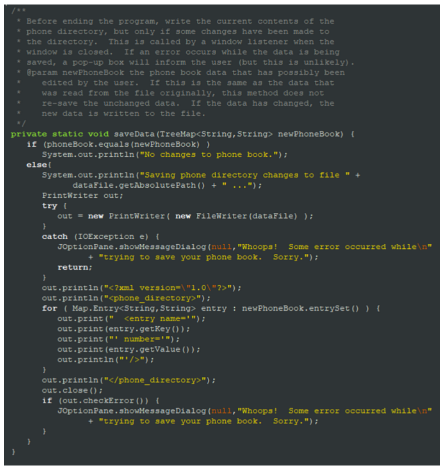
4.4.2 LightGBM catches context with higher frequence.

The LightGBM model did not perform well on most low frequency concepts even compare to linear predictor, while it out performed the linear predictor especially on high frequency concepts.

However, in some cases, the LightGBM outperformed neural network even for low frequency concepts. By analyzing different cases, the following case in Figure 9 identifying concept "type: TreeMap<String, String>" was representative.

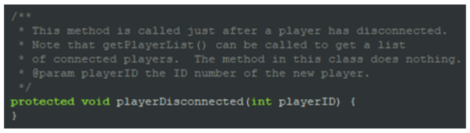
In this example, the code tried to record a phone directory to the file system. In comment, the data type of phone directory was not clearly identified, so the neural network did not detect the concept "type: TreeMap<String, String>", while LightGBM successfully detects this concept according to the context in training comment.

On the other hand, neural network incorrectly identified 3 other code pieces as relevant to the concept, while LightGBM only had one



```
/**
 * Before ending the program, write the current contents of the
 * phone directory, but only if some changes have been made to
 * the directory.  This is called by a window listener when the
 * window is closed.  If an error occurs while the data is being
 * saved, a pop-up box will inform the user (but this is unlikely).
 * @param newPhoneBook the phone book data that has possibly been
 *    edited by the user.  If this is the same as the data that
 *    was read from the file originally, this method does not
 *    re-save the unchanged data.  If the data has changed, the
 *    new data is written to the file.
 */
private static void saveData(TreeMap<String,String> newPhoneBook) {
   if (phoneBook.equals(newPhoneBook) )
      System.out.println("No changes to phone book.");
   else{
      System.out.println("Saving phone directory changes to file " +
            dataFile.getAbsolutePath() + " ...");
      PrintWriter out;
      try {
         out = new PrintWriter( new FileWriter(dataFile) );
      }
      catch (IOException e) {
         JOptionPane.showMessageDialog(null,"Whoops!  Some error occurred while\n"
               + "trying to save your phone book.  Sorry.");
         return;
      }
      out.println("<?xml version=\"1.0\"?>");
      out.println("<phone_directory>");
      for ( Map.Entry<String,String> entry : newPhoneBook.entrySet() ) {
         out.print("  <entry name='");
         out.print(entry.getKey());
         out.print("' number='");
         out.print(entry.getValue());
         out.println("'/>");
      }
      out.println("</phone_directory>");
      out.close();
      if (out.checkError()) {
         JOptionPane.showMessageDialog(null,"Whoops!  Some error occurred while\n"
               + "trying to save your phone book.  Sorry.");
      }
   }
}
```

Figure 9: Example of LightGBM outperforms.



```
/**
 * This method is called just after a player has disconnected.
 * Note that getPlayerList() can be called to get a list
 * of connected players.  The method in this class does nothing.
 * @param playerID the ID number of the new player.
 */
protected void playerDisconnected(int playerID) {
}
```

Figure 10: Example of linear predictor outperforms.

incorrect positive identification. In another word, LightGBM was stricter in false-positive cases, which was the reason that LigthGBM outperforms neural network on this concept.

4.4.3 *Linear predictor restricts false-positive.*

In linear predictor, the deeper logic was not involved in comment semantic detection, the only feature used was the connection between terms and concepts. On the other hand, neural network used deep logic frequently, which was considered as the advantage of neural network in most cases. However, the usage of deep logic also brought false-positive. There were cases that concepts were not involved in the code of a comment but identified by neural network.

In the following case, the concept "MethodCallExpr: sleep" was not involved since there was no content in the method. However, since there were phrases like "does nothing" and "disconnected" in the comment, the neural network determines the concept "MethodCallExpr: sleep" as involved. However, since linear predictor built up a weaker connection without considering deep logic and combinations of terms, it did not classify this concept as positive.

5 CONCLUSIONS

In this work, we first proposed a set of methods to extract concepts from programming code and connect code pieces with corresponding annotations. Then multiple predictor models were conducted to capture the connection between annotation semantics and code concepts, and predict concepts given annotation as natural language document.

In the experiment, all models were compared with a random guess model as baseline. The result showed all models outperform the baseline. The neural network model performed the best among the models, and lightGBM performed better than linear predictor when a concept had higher frequency, or a document included more concepts. In specially cases, LightGBM and linear predictor can outperform neural network since they did not have false-positive classifications as much as neural network. Although neural network outperformed other models in most cases, it consumed much more time, which should be considered in applications as trade-off. These research would shed light on automatic programmer learner assistant such as example code recommendation and auto coder.

REFERENCES

[1] John R Anderson, C Franklin Boyle, Albert T Corbett, and Matthew W Lewis. 1990. Cognitive modeling and intelligent tutoring. *Artificial intelligence* 42, 1 (1990), 7–49.

[2] Robert K Atkinson, Sharon J Derry, Alexander Renkl, and Donald Wortham. 2000. Learning from examples: Instructional principles from the worked examples research. *Review of educational research* 70, 2 (2000), 181–214.

[3] Cory J Butz, Shan Hua, and R Brien Maguire. 2004. A web-based intelligent tutoring system for computer programming. In *Proceedings of the 2004 IEEE/WIC/ACM international conference on web intelligence.* IEEE Computer Society, 159–165.

[4] Anna Corazza, Valerio Maggio, and Giuseppe Scanniello. 2015. On the coherence between comments and implementations in source code. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on.* IEEE, 76–83.

[5] Starr Roxanne Hiltz and Barry Wellman. 1997. Asynchronous learning networks as a virtual classroom. *Commun. ACM* 40, 9 (1997), 44–49.

[6] Matthew J Howard, Samir Gupta, Lori Pollock, and K Vijay-Shanker. 2013. Automatically mining software-based, semantically-similar words from comment-code mappings. In *Proceedings of the 10th Working Conference on Mining Software Repositories.* IEEE Press, 377–386.

[7] I-Han Hsiao and Piyush Awasthi. 2015. Topic facet modeling: semantic visual analytics for online discussion forums. In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge.* ACM, 231–235.

[8] I-Han Hsiao and Yi-Ling Lin. 2017. Enriching programming content semantics: An evaluation of visual analytics approach. *Computers in Human Behavior* 72 (2017), 771–782.

[9] Wen Hua, Zhongyuan Wang, Haixun Wang, Kai Zheng, and Xiaofang Zhou. 2017. Understand short texts by harvesting and analyzing semantic knowledge. *IEEE transactions on Knowledge and data Engineering* 29, 3 (2017), 499–512.

[10] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems.* 3146–3154.

[11] Päivi Kinnunen and Lauri Malmi. 2006. Why students drop out CS1 course?. In *Proceedings of the second international workshop on Computing education research.* ACM, 97–108.

[12] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. *Acm Sigcse Bulletin* 37, 3 (2005), 14–18.

[13] Yihan Lu and I-Han Hsiao. 2017. Personalized Information Seeking Assistant (PiSA): from programming information seeking to learning. *Information Retrieval Journal* 20, 5 (2017), 433–455.

[14] Yihan Lu and I-Han Hsiao. 2018. Modeling Semantics between Programming Codes and Annotations. In *Proceedings of the 29th on Hypertext and Social Media.* ACM, 101–105.

[15] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014), 51–61.

[16] Valerio Maggio. 2015. Comments and Implementations—A Public Benchmark. (2015). http://www2.unibas.it/gscanniello/coherence

[17] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. 2012. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* ACM, 153–160.

[18] Jan L Plass, Roxana Moreno, and Roland Brünken. 2010. *Cognitive load theory.* Cambridge University Press.

[19] Shashank Singh. 2017. *CodeReco-A Semantic Java Method Recommender.* Ph.D. Dissertation. Arizona State University.

[20] Lin Tan, Yuanyuan Zhou, and Yoann Padioleau. 2011. aComment: mining annotations from comments and code to detect interrupt related concurrency bugs. In *Software Engineering (ICSE), 2011 33rd International Conference on.* IEEE, 11–20.