Domain Model Discovery from Textbooks for Computer Programming Intelligent Tutors

Rabin Banjade, Priti Oli, Lasang Jimba Tamang, Jeevan Chapagain, Vasile Rus
Department of Computer Science, Institute of Intelligent Systems
University of Memphis, Memphis, TN, USA
{rbnjade1, poli, ltamang, cjeevan, vrus} @memphis.edu

Abstract

We present a novel approach to intro-to-programming domain model discovery from textbooks using an over-generation and ranking strategy. We first extract candidate key phrases from each chapter in a Computer Science textbook focusing on intro-to-programming and then rank those concepts according to a number of metrics such as the standard tf-idf weight used in information retrieval and metrics produced by other text ranking algorithms. Specifically, we conduct our work in the context of developing an intelligent tutoring system for source code comprehension for which a specification of the key programming concepts is needed - the system monitors students' performance on those concepts and scaffolds their learning process until they show mastery of the concepts. Our experiments with programming concept instruction from Java textbooks indicate that the statistical methods such as KP Miner method are quite competitive compared to other more sophisticated methods. Automated discovery of domain models will lead to more scalable Intelligent Tutoring Systems (ITSs) across topics and domains, which is a major challenge that needs to be addressed if ITSs are to be widely used by millions of learners across many domains.

Introduction

Domain modeling is the task of specifying the units of knowledge, also called Knowledge Components (KCs), in a target domain such as physics, biology, or computer programming. A domain model includes a structure that specifies the relationship among the KCs, typically in the form of a prerequisite knowledge structure suggesting a specific trajectory towards mastery, i.e., a particular order in which students should master the KCs (Goldin, Pavlik Jr, and Ritter 2016; Koedinger, Corbett, and Perfetti 2012; Chau et al. 2020). Domain model can also link the KCs to specific learning activities or objects that allow learners to master those KCs through practice. We can make an argument that domain modeling should be expanded to include all key concepts, skills, ideas, principles, other types of knowledge such as procedures and processes, and the values, identity, and epistemology of the community of experts or professionals active in the target domain. That is, if the goal of instruction is to prepare a successful expert in a domain, besides the KCs in textbooks, a learner must learn, for instance, the values of the experts in the target domain and therefore domain models must specify those additional aspects of becoming an expert in a community of experts. However, a broader discussion about what a domain model is or should be is beyond the scope of this paper.

In this work, our goal is to automatically discover a standard domain model by extracting key concepts or KCs from textbooks. Our work is done in the context of developing and investigating the effectiveness of an ITS for source code comprehension. A typical ITS works with students through various instructional activities to help them master key concepts of a target domain. The underlying domain model guides the functionality of ITSs and has a major impact on the system's effectiveness to induce learning gains and on the overall student learning experience.

The key concepts in a target domain that students need to master are often specified by experts such as domain experts, pedagogical experts, and ITS designers. This expert-driven approach is tedious, expensive, time-consuming, and makes ITS development hard to scale across domains. Furthermore, expert-defined domain models can be error prone or inadequate for instructional purposes as "experts may forget the difficulties that novice learners face." (Goldin, Pavlik Jr, and Ritter 2016; Koedinger, Corbett, and Perfetti 2012). This can have negative consequences on assessing learners' knowledge state, which leads to poor adaptivity of ITSs and, consequently, a negative impact on the effectiveness and overall quality of the provided instruction.

To overcome the above-mentioned challenges, there is a need for automated or semi-automated methods. In this paper, we investigate and propose a novel automated method for domain model discovery, particularly focusing on computer programming textbooks. Such an automation has several advantages. First, it relieves the need for handpicking key concepts as the textbook authors already put much effort in doing so. Second, it helps discover the ordering of KCs necessary for tutoring systems as textbooks present the KCs in a particular order (which could be refined based on, for instance, student performance data). Third, automating knowledge discovery from the textbooks will save a lot of time and effort for tutoring system developers. In particular, it will help with porting an ITS platform from one domain to another more easily thus leading to more scalable ITSs across topics and domains.

Our approach to automatically extract domain models from textbooks is to rely on keyphrase extraction methods to identify a domain's KCs. The problem of extracting KCs from a Computer Science textbook poses several unique challenges and opportunities. For instance, Computer Science textbooks contain domain-specific words such as *for* to describe the concept of loops, and therefore this keyphrase requires special handling to distinguish it from the regular preposition *for*. Furthermore, Computer Science textbooks contain many code examples and their plain text explanations, so, there is a practical need for distinction between the two. Typical keyphrase extraction methods work primarily on pure text. This combination of code and text in Computer Science textbooks is also a great opportunity for domain modeling as it facilitates the linking of KCs to specific learning activities such as code comprehension activities. For instance, a Java code example in an intro to Java programming textbook can be linked to the key concepts it covers by inspecting the KCs mentioned in the explanatory text. Furthermore, intro to programming textbooks document major misconceptions students exhibit while learning programming. Our goal is to expand a typical domain model with the key misconceptions students have, critical for feedback opportunities in ITSs. In sum, our work on automated discovery of domain modeling addresses the following four key tasks: (1) knowledge component extraction, (2) prerequisite knowledge structure discovery (3) linking of KCs to learning objects/activities, which in our case, a knowledge object is a Java example in the textbook, and (4) misconception extraction.

The outline of the paper is as follows. The next section, *Related Work* briefly highlights key prior efforts in the areas of automated extraction of domain models and the related area of automatic domain model refinement as well as previous efforts on general techniques for keyword or keyphrase extraction from texts. *The Approach* section outlines the key steps of the proposed approach to domain modeling extraction from intro-to-programming textbooks. The following section presents details about the experiments we conducted and the results obtained. The *Conclusions* section summarizes the contributions of the paper and outlines future work.

Related Work

In this section, we briefly review prior efforts related to automated extraction of domain models and the related area of automatic domain model refinement and previous efforts on general techniques for keyword or keyphrase extraction from texts.

When developing domain models, there are three significant information sources: experts, textbooks (written by domain experts), and learner data. We will briefly review work, focusing primarily on extracting or refining domain models from data (text or structured data). For instance, student performance data is often used as input to domain modeling methods in the form of a Q-matrix linking knowledge components to instructional items in a domain, such as solutions to problems, steps in a solution, or a student explanation. Such Q-matrices are useful primarily for well-defined domains and less so for ill-defined domains (Goldin, Pavlik Jr, and Ritter 2016). Given such a Q-matrix, one can infer a set of latent variables that can partition a set of instructional items based on learner responses to those items. Prediction of student performance based on the discovered latent skills is used to evaluate the inferred domain model. There are several issues with such approaches to domain model discovery: (1) interpreting what skills the latent variables represent and (2) the need for student performance data. The latter is quite challenging when developing domain models for emerging domains such as data science or nanotechnology for which student data may not yet be available. Often Q-matrix-based approaches start with a domain model which is another challenge as they require some other source for the start domain model. The main goal is such cases is to refine the start domain model based on student performance data, i.e., discovering a new set of skills in the form of latent variables that best predict student performance.

Extracting KCs from textbooks has been explored before for the domain of information retrieval. For instance, Chau and colleagues (Chau et al. 2020) adopted a supervised machine learning approach based on a set of expert-defined features. The features they used fall into three broad categories: linguistic, positional, or statistical. Similar to our work, they use an over-generation and ranking approach. They first generated a large set of candidate keyphrases and then applied a selection criterion or filter to rank and detect the true domain concepts. Unlike us, they use a part-of-speech tagger to identify noun phrases together with a set of regular expressions. They retain only noun phrases of up to 4 tokens as candidate keyphrases. As we explain later, the use of a part-of-speech tagger does not provide good results for Computer Science textbooks. We do consider candidate phrases as having up to 4 tokens similar to them but we do not limit ourselves to only noun phrases as other phrases can indicate key programming concepts such as *sorting*. They compared their approach to a number of baseline methods and some off-the-shelf algorithms such as TextRank (Mihalcea and Tarau 2004), which we have used and reported in this work as well.

The extraction of keyphrases from text has been explored quite extensively for various purposes. (Dominowska and Ragno 2009) used keyphrase extraction from query logs to create a content-to-keyphrase index. Similarly, (Zhang, Zincir-Heywood, and Milios 2005) investigated various keyphrase extraction algorithms in the context of Web document corpora. They compared three different algorithms TF-IDF, KEA, and key term, and showed that narrative text classification could significantly improve keyphrase extraction. Keyphrase extraction is also useful for generating questions from documents. (Subramanian et al. 2017) used a two-stage framework to generate questions from documents using extracted keyphrases. Furthermore, keyphrase extraction was applied to different kinds of text: news articles (Marujo et al.), scientific articles (Nguyen and Kan 2007), medical documents (Sarkar 2009), or online policies (Audich, Dara, and Nonnecke 2016).

There are basically two types of keyphrase extraction approaches: supervised, and unsupervised. We focus on unsupervised approaches. One of the well-known and widely used unsupervised algorithms in keyphrase extraction relies on the TF-IDF weighting method, which uses term frequency and inverse document frequency to rank keyphrases (Salton, Wong, and Yang 1975). (Witten et al. 2005) used a classifier based on Bayes' theorem to classify words in documents (preferably from the same domain) as being a keyword/keyphrase or not. (El-Beltagy and Rafea 2009) proposed the KP-Miner algorithm, a variant of the TF-IDF algorithm which prioritizes multiword key phrases by introducing a boosting factor and considering the number of documents equal to 1 for IDF calculation for such terms. RAKE (Rose et al. 2010) is a statistical unsupervised keyphrase extraction algorithm based on the observations that keywords frequently contain multiple words with standard punctuation or stop words, i.e., functioning words like 'and,' 'of,' 'the,' etc. with minimum lexical meaning. YAKE (Campos et al. 2020) is an online keyword extraction tool which builds upon statistical text features extracted from a single document to identify and rank the most important keywords.

Apart from the statistical methods mentioned, another class of unsupervised key phrase extraction algorithms focuses on co-occurrence analysis. A co-occurrence graph is generated where nodes are key phrases, and links indicate whether they co-occur, e.g., within a span of text whose size can be controlled. (Mihalcea and Tarau 2004) proposed a graph-based keyword extraction approach. Their TextRank algorithm generates a graph based on a co-occurring analysis of words in a window of a given length (e.g., 3-5 tokens). The PageRank (Page et al. 1999) algorithm is then applied on the co-occurrence graph to distinguish the important nodes or phrases in the graph. Similarly, TopicRank (Bougouin, Boudin, and Daille 2013) is a graph-based keyphrase extraction algorithm in which nodes represent topics that consist of sets of candidate terms clustered around shared sub-terms. SingleRank (Wan and Xiao 2008) is also a graph-based keyword extraction algorithm that ranks phrases in the text based on their word weights. SingleRank is an extension of TextRank by assigning weights to the graph edges. We applied those graph-based algorithms to our data and report results for comparison purposes.

The Approach

We adopted an over-generation and ranking approach to discover the KCs of the intro to programming domain. Our inputs are Computer Science textbooks. In particular, the unit of processing are chapters in such books. For instance, a document is a chapter for computing TF-IDF weights. We have experimented with the following unsupervised methods: TF-IDF, KP-Miner and YAKE (statistical methods) and TextRank, TopicRank, and SingleRank (graph-based methods). To the best of our knowledge, domain model discovery for intro to computer programming from textbooks has not been studied before. As a result, there is no benchmark dataset for this problem.

There are several advantages of using textbooks to extract domain models. First, textbooks describe a target domain's knowledge with an instructional purpose in mind. The authors of textbooks spend significant efforts to define the key concepts, present them in a specific order, and provide plenty of instructional activities to practice those concepts. Furthermore, the textbooks' structure in chapters and sections facilitates the extraction of key concepts using, for instance, statistical methods. It enables the organization of the extracted key concepts in more complex structures such as prerequisite knowledge structures and taxonomies.

As already noted, intro-to-programming textbooks have a peculiarity in that they contain both code examples and related explanatory text. Since our main objective here is to extract the KCs, we focus only on the text explanations instead of code examples. The code examples generally contain comments in text form that explain the code as well. Often, those comments repeat concepts described in the surrounding explanatory text as well and are therefore redundant for our purposes. It is possible to extract more abstract concepts directly from code, e.g., by performing a static syntax analysis of the code but is beyond the scope of this paper. It should be noted that there is a major disadvantage of such methods - the extracted concepts are harder to interpret. For these reasons, we focus here primarily on extracting the text portions of intro-to-programming textbooks. To extract the descriptive text from textbooks, we developed a Naive Bayes classifier that can classify each line in textbooks as either explanatory text or code. This classifier had a classification accuracy of 94% with F-score of 0.9. The explanatory text, thus extracted is used for further analysis. We used the following textbook *Introduction to JAVA programming* (Liang 2011) for our experiments.

Annotation of Key Concepts

In order to evaluate the performance of the proposed methods for key concept extraction, we needed to create a gold standard of such KCs, i.e., a set of true concepts related to intro to programming. To best of our knowledge, there is no standard benchmark readily available and therefore, we created a gold standard ourselves. We started with the key concepts at the end of the chapters that the author of the Introduction to JAVA Programming textbook selected. The list of concepts at the end of each chapter are not complete and thus we manually extracted all the key concepts from two sample chapters(*loops* and *sorting*). We recruited two graduate students in Computer Science with expertise in computer programming and who were trained on and followed a similar annotation coding procedure to the one described in (Wang et al. 2020), except that the guidelines were not restricted to consider only noun phrases as key concepts. Our annotators separately annotated the key concepts in each chapter and then discussed the annotated concepts with each other to account for missing key concepts and solve any disagreements. For the initial set of key concepts extracted by each author, the inter-annotator agreement, as measured by Kappa statistic, was $\kappa = 0.88$.

Candidate Concept Generation

We process each chapter as an input document for candidate extraction in our textbook, there are 33 chapters. All n-grams ($n = 1...4$) are considered candidate keyphrases. We used this n-gram range for extracting candidate key concepts based on a quick analysis of the textbook by ourselves as well as based on literature, e.g., Chau and colleagues (Chau et al. 2020) define *domain concepts* as "single words or short phrases of two to four words." The candidate n-grams contain stop words and punctuation marks. We do not remove stop words for candidate key phrase selection because important key concepts might contain stop words as in *continuation of loop* or *pre-test and post-test loop*. Furthermore, some key phrases like 'for loop', 'while loop' contain stop words 'for' and 'while' which are important key concepts in computer programming.

We do not filter candidate key phrases based on the Part of Speech (POS) sequence of candidate keywords because ready-to-use taggers became inadequate for keyphrases such as *for loop, while loop, or merge sort* and special POS taggers are not available for computer programming related texts.

Ranking of Candidate Keywords

We rank candidate keyphrases from our previous step using statistical and graph based methods as described next.

Statistical Methods For Key Concept Extraction

- **TFIDF**: In information retrieval, TFIDF is used as a way to quantify how important each word/token is for a particular document for retrieval purposes, i.e., with respect to being able to distinguish that document from others for retrieval purposes. Also, due to its relative simplicity and interpretability it can also serve as a solid baseline. Specifically, TF-IDF assigns each candidate keyphrase a weight based on the following formula:

  $$TFIDF = tf * idf$$

  where, tf = term frequency and idf = inverse document frequency.

  In our case, a higher TFIDF score means the keyphrase occurs a lot in the current document/chapter (high term frequency) and not so much in other documents/chapters (high inverted document frequency). For instance, the chapter of loops will refer to loops a lot (high term frequency) whereas many other chapters will do less so (high inverted document frequency).

- **KP-Miner:** The KP-Miner algorithm is based on assumption that occurrences of keyphrases is much less frequent than the occurrence of single terms within the same document. It uses a boosting factor for compound terms in order to remove bias, e.g., imposed by the TF-IDF method. The KP-Miner algorithm computes a weight for each candidate keyphrase based on the following equation:

  $$w_{ij} = tf_{ij} * idf * B_i * P_f$$

  where, w_{ij} = weight of term t_j in Document D_i

  tf_{ij} = frequency of term t_j in Document D_i

  idf = $log_2 N/n$ where N is the number of documents in the collection and n is number of documents where term t_j occurs at least once. If term is a compound term, n is set to 1.

  B_i = boosting factor associated with document D_i

  P_f = the term position associated factor. If position rules are not used this is set to 1

- **YAKE:** YAKE considers phrases that do not begin and end with stop words. YAKE makes use of five features for candidate keyphrase ranking: casing (gives more importance to capitalized words and acronyms), word position (gives more importance to words at the beginning of the document), word frequency (frequency of terms), word relatedness to context (qualifies word relatedness based on words present on the left and right side vicinity) and word different sentence (quantifies how often a word appears within different sentences).

Graph-Based Methods For Ranking of Key Words

- **TextRank:** TextRank sorts candidate keyphrases by generating first an undirected and unweighted graph - an edge between words is created if the words represented in the corresponding graph nodes co-occur within a window of M words. The PageRank (Page et al. 1999) algorithm is then run on the graph to compute a score for each node V_i based on the following equation:

  $$S(V_i) = (1 - \lambda) + \lambda * \sum_{j \epsilon N(V_i)} \frac{1}{N(V_j)} S(V_j)$$

  where $N(V_i)$ represents co-occurring terms of V_i, $N(V_j)$ represents neighbours of V_j and λ is the probability of transition between nodes.

- **SingleRank:** SingleRank extends the TextRank algorithm by incorporating weights for the edges. An edge weight is equal to the number of co-occurrences of the two words. After computing a scoring function similar to TextRank, the constituent words' score is summed up and top-ranking keywords are returned.

- **TopicRank:** For TopicRank, the preprocessed candidate keyphrases are grouped into different topics using a hierarchical agglomerative clustering method (Papagiannopoulou and Tsoumakas 2020). A graph of topics is created and edges are weighted based on phrases' offset positions in the text. TextRank is then used to rank the topics and a keyphrase candidate is selected from each of the N most important topics.

Evaluation and Results

We conducted a set of experiments using the over-generation and ranking approach based on the ranking methods outlined above. For each method, we extracted KCs and compared them to the key concepts extracted by our annotators from two sample chapters in the textbooks, as mentioned earlier. However, it should be noted that all the chapters from the textbook were used to extract KCs.

We evaluated the KCs extraction in two ways. First, we did an exact match between the KCs produced by any of the above mentioned methods and the gold standard KCs and report average precision, recall, and F1 score for the top 100 KCs extracted by each algorithm - see table 1. These three measures are standard for the assessment of key phrases. evaluation (Boudin 2016). It should be noted that we used rank 100 for this evaluation to be able to report precision and recall while at the same time use a reasonable number of candidate KCs. The gold standard for the two chapters contain 37-52 concepts only.

As a second evaluation approach, we computed relaxed precision (Elhadad et al. 2015) at rank k = 10, 20, 30, 50. We labeled each candidate key phrase as a key concept if there is any word overlap between the predicted key phrase and the gold standard phrase (both in the case of continuous and discontinuous spans). We also analyzed if the meaning conveyed by the gold standard and extracted KC is similar. For example, we deemed *merge sort algorithm* as a KC although the gold standard specifies *merge sort* as a KCs. The reason is because both refer to the same key concept i.e. *merge sort algorithm.* We show the results for relaxed precision at different ranks in table 2 where we also show recall along with relaxed precision.

Algorithm	Precision	Recall	F1 score
TF IDF	0.10	0.44	0.16
KP-Miner	0.14	0.60	0.23
YAKE	0.11	0.47	0.18
TextRank	0.07	0.29	0.11
TopicRank	0.09	0.37	0.14
SingleRank	0.13	0.55	0.21

Table 1: Average precision and recall at k = 100

k	5 P/R	10 P/R	20 P/R	30 P/R	50 P/R
TF-IDF	.6/.07	.65/.15	.65/.23	.5/.23	.38/.26
KP-Mnr	.6/.12	.75/.17	.65/.28	.60/.34	.44/.42
YAKE	.7/.12	.6/.17	.6/.17	.52/.25	.36/.34
Text-R	.7/.02	.6/.08	.55/.14	.65/.14	.42/.19
Topic-R	.6/.08	.4/.12	.30/.17	.35/.17	.3/.21
Single-R	.6/.12	.7/.17	.60/.25	.6/.30	.46/.34

Table 2: Average relaxed precision (P) and recall (R) at k = 5, 10, 20, 30, 50

At a closer look of the results, in particular those shown in table 1, we note that statistical methods outperform graph-based methods for KC extraction for the intro to programming domain. This could be an effect of the way we calculated precision and recall based on exact match and the fact that most of the key concepts in computer programming are n-grams where n>1. The better performance of KPMiner might be attributed to the fact that for n-grams with n>1, it considers document frequency as 1, giving more importance to multi-word key concepts. Also worth considering,

TF IDF: ***loop***, number, body, ***loop body***, variable, following, condition, display, ***iteration***, value, ***loop continuation condition***
KP MINER: ***loop***, write program, ***loop body***, following, write, continuation condition, loop continuation, ***loop continuation condition***, chapter loop, prime number, ***while loop***
YAKE: ***loop continuation condition,*** loop chapter loops, numbers write program, ***loop body,*** number, enter, chapter loop, loop continuation, loop chapter loop, loop body loop, write loop
TextRank: ***loop***, loop end loop, syntax of for loop, write nested ***for loop,*** following loop, checking loop control, program, example of while loop, ***nested for***, using for-loop
TopicRank: ***loop***, number, user, ***loop body***, statement, input, string, example, next guess, year
SingleRank: ***loop,*** syntax of for loop, loop body loop, ***pretest loop, posttest loop,*** following loop, use break in loop, used in loop statement

Table 3: Top 10 ranked KCs using the various methods. True KCs are shown in italic bold

TF-IDF weighs terms unique to a particular chapter higher (due to higher IDF) which could be one of the potential reasons for better performance for the task of domain modeling. For example: *loop continuation condition* is only mentioned and explained in the chapter on loops out of the whole textbook. Graph-based methods do not seem to perform as well as the statistical methods, as shown in table 1. There is one exception - the SingleRank method. This could potentially be due to the co-occurrence frequency consideration by the SingleRank algorithm - key concepts frequently occur in a particular chapter. If we consider the results in table 2, we notice that the graph-based algorithms are on par with statistical algorithms. We notice that graph-based algorithms tend to select key phrases highly relevant to a given chapter, but many of these cannot be considered key concepts for intro-to-programming domain modeling. One such example is the 'syntax of for loop' phrase which is one of the top-ranked key phrases by the graph-based algorithms. We consider it a key concept for relaxed precision because of the presence of the keyword loop, but we do not consider it a key concept for standard precision and recall calculation.

Table 3 shows the top 10 ranked KCs for one of the chapters in the textbooks for each of the methods we experimented with.

Conclusions

In this paper, we evaluated statistical and graph-based methods for domain model extraction and presented the results of applying those methods to extract KCs for the target domain of intro to computer programming. The results suggest that unsupervised key phrase extraction methods can be used for domain model discovery from Computer Science textbooks. We plan to extend this work by inferring a prerequisite knowledge structure as well as to link instructional tasks to the KCs extracted. Such automated discovery of domain models will lead to more scalable ITSs across topics and domains.

Acknowledgement

This work has been supported by two NSF awards: the Learner Data Institute (NSF; award 1934745) and CSEdPad: Investigating and Scaffolding Students' Mental Models during Computer Programming Tasks to Improve Learning, Engagement, and Retention (NSF award 1822816). The opinions, findings, and results are solely the authors' and do not reflect those of NSF.

References

Audich, D. A.; Dara, R.; and Nonnecke, B. 2016. Extracting keyword and keyphrase from online privacy policies. In *2016 Eleventh International Conference on Digital Information Management (ICDIM)*, 127–132. IEEE.

Boudin, F. 2016. pke: an open source python-based keyphrase extraction toolkit. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, 69–73.

Bougouin, A.; Boudin, F.; and Daille, B. 2013. Topicrank: Graph-based topic ranking for keyphrase extraction.

Campos, R.; Mangaravite, V.; Pasquali, A.; Jorge, A.; Nunes, C.; and Jatowt, A. 2020. Yake! keyword extraction from single documents using multiple local features. *Information Sciences* 509:257–289.

Chau, H.; Labutov, I.; Thaker, K.; He, D.; and Brusilovsky, P. 2020. Automatic concept extraction for domain and student modeling in adaptive textbooks. *International Journal of Artificial Intelligence in Education* 1–27.

Dominowska, E., and Ragno, R. 2009. Key phrase extraction from query logs. US Patent 7,577,643.

El-Beltagy, S. R., and Rafea, A. 2009. Kp-miner: A keyphrase extraction system for english and arabic documents. *Information systems* 34(1):132–144.

Elhadad, N.; Pradhan, S.; Gorman, S.; Manandhar, S.; Chapman, W.; and Savova, G. 2015. Semeval-2015 task 14: Analysis of clinical text. In *proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 303–310.

Goldin, I.; Pavlik Jr, P. I.; and Ritter, S. 2016. Discovering domain models in learning curve data. *Design Recommendations for Intelligent Tutoring Systems* 115.

Koedinger, K. R.; Corbett, A. T.; and Perfetti, C. 2012. The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive science* 36(5):757–798.

Liang, D. 2011. Introduction to java programming.

Marujo, L.; Gershman, A.; Carbonell, J.; Frederking, R.; and Neto, J. P. Supervised topical key phrase extraction of news stories using crowdsourcing, light filtering and co-reference normalization.

Mihalcea, R., and Tarau, P. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 404–411.

Nguyen, T. D., and Kan, M.-Y. 2007. Keyphrase extraction in scientific publications. In *International conference on Asian digital libraries*, 317–326. Springer.

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Papagiannopoulou, E., and Tsoumakas, G. 2020. A review of keyphrase extraction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10(2):e1339.

Rose, S.; Engel, D.; Cramer, N.; and Cowley, W. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory* 1:1–20.

Salton, G.; Wong, A.; and Yang, C.-S. 1975. A vector space model for automatic indexing. *Communications of the ACM* 18(11):613–620.

Sarkar, K. 2009. Automatic keyphrase extraction from medical documents. In *International Conference on Pattern Recognition and Machine Intelligence*, 273–278. Springer.

Subramanian, S.; Wang, T.; Yuan, X.; Zhang, S.; Bengio, Y.; and Trischler, A. 2017. Neural models for key phrase detection and question generation. *arXiv preprint arXiv:1706.04560*.

Wan, X., and Xiao, J. 2008. Single document keyphrase extraction using neighborhood knowledge. In *AAAI*, volume 8, 855–860.

Wang, M.; Chau, H.; Thaker, K.; Brusilovsky, P.; and He, D. 2020. Concept annotation for intelligent textbooks. *arXiv preprint arXiv:2005.11422*.

Witten, I. H.; Paynter, G. W.; Frank, E.; Gutwin, C.; and Nevill-Manning, C. G. 2005. Kea: Practical automated keyphrase extraction. In *Design and Usability of Digital Libraries: Case Studies in the Asia Pacific*. IGI global. 129–152.

Zhang, Y.; Zincir-Heywood, N.; and Milios, E. 2005. Narrative text classification for automatic key phrase extraction in web document corpora. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*, 51–58.