An Initial Exploration of Machine Learning Techniques to Classify Source Code Comments in Real-time

Ms. Phyllis Beck, Mississippi State University
Dr. Mahnas Jean Mohammadi-Aragh, Mississippi State University

Dr. Jean Mohammadi-Aragh is an assistant professor in the Department of Electrical and Computer Engineering at Mississippi State University. Dr. Mohammadi-Aragh investigates the use of digital systems to measure and support engineering education, specifically through learning analytics and the pedagogical uses of digital systems. She also investigates fundamental questions critical to improving undergraduate engineering degree pathways. . She earned her Ph.D. in Engineering Education from Virginia Tech. In 2013, Dr. Mohammadi-Aragh was honored as a promising new engineering education researcher when she was selected as an ASEE Educational Research and Methods Division Apprentice Faculty.

Christopher Archibald, Mississippi State University

An Initial Exploration of Machine Learning Techniques to Classify Source Code Comments

Abstract

Providing real-time feedback to novice programmers is critical to their ability to learn to program. Higher enrollment in introductory computer science courses reduces the amount of time for individual student-instructor interaction. Reduced interaction time equates to a reduction in the time for and amount of instructor feedback. Building on our work involving manual classification and analysis of student source code comments, in this full paper we explore how machine learning techniques can be leveraged to provide automated feedback to students with regards to their computational thinking processes. This paper discusses the initial classification of student source code comments using supervised machine learning methods. In this phase of classification, we focus on whether a comment is *sufficient* or *insufficient*. The classification process is broken down into three steps: text processing, data exploration, and comment classification using the Multinomial Naïve Bayes Classifier and a Random Forest Classifier.

We detail the text processing requirements, including how to prepare the raw student data using natural languages processing techniques such as stop word filtration, tokenization, and lemmatization. We also show how the data preparation process can affect the final classification outcome. Using Multinomial Naïve Bayes we achieved a precision rate of 82%. Using a Random Forest classifier and lemmatization we achieved a classification precision of 90%. We conclude with a description of how the current classification results can be used to provide real-time feedback to students while they are learning to program. Towards our ultimate goal of providing comprehensive real-time feedback to students, we describe future research plans, which include using unsupervised machine learning techniques to move beyond basic binary classification.

1. Introduction

In this paper, we explore the process for training two supervised machine learning classification algorithms to classify student code comments as *sufficient* or *insufficient* using Multinomial Naive Bayes Classifier and a Random Forest Classifier. We are classifying comments from student lab submissions as part of a larger NSF funded writing-to-learn to program project in which we are developing a framework for allowing students to self-monitor and self-assess their own metacognition [1,2]. Students are provided with an Integrated Development Environment (IDE) that allows the students to use Restructured Text in conjunction with Python code to simultaneously produce their lab code and report. Traditional lab students use Python's Idle editor and produce a lab report in a separate document. The basis for this paper begins with the development of our qualitative codebook, developed as part of previous research efforts using a mixed methods investigation.

This paper focuses on a single section of the codebook where a single comment is classified as either *sufficient* or *insufficient*. This is the first level of classification. An *insufficient* comment is one that is too short to warrant a more complex classification or does not provide any additional

insight into the function or purpose of the code. In contrast, a *sufficient* comment displays adequate writing and further clarifies and improves the understanding the source code from an outside perspective. A second level of classification is still needed to further classify *sufficient* comments into their own categories. These include conceptual, reflective, organizational and literal comment types. This classification system is further documented in a previous work [3].

2. Cleaning and Processing the Data

The data set for this investigation comes from six sections of an Introduction to Programming course; two sections (section 05 and section 08) are writing-to-learn to program sections and the additional four sections are taught using the traditional lab approach. The training set consists of 761 comments with 30% randomly sampled out as the test data and the remaining 70% serves as the training data. All students were taught by the same course instructor but labs were taught with different teaching assistants. Both of the writing-to-learn sections were taught by the same teaching assistant to help maintain consistency.

	label	comment	raw_length
0	0	Explain the code to the user	28
1	1	Import math for the functions you will use later	48
2	1	Ask for the length of the tank, in meters	41
3	1	Ask for the diameter of the water tank, in meters	49
4	1	Ask for the number of residents the tank will ...	51
5	1	Each person uses a constant 2 liters of water ...	53
6	0	Start the table.	16
7	1	Divide the diameter by 2 to find the radius	43
8	1	Calculate the volume when no water has been us...	69
9	1	Calculate the days remaining before the water ...	54

Figure 1: Raw Labeled Data

During the course of the semester, students completed nine labs in total, eight of the nine labs will be used in this data set. Lab 7 is a two-part lab and Lab 9 was a top-down design project that was not accessible for evaluation. The structure of the labs is as follows, students are paired into groups of two and there are typically 24 students per a lab section. Students are required to complete the lab using pair programming and they are required to complete the lab and report and then demo it to the teaching assistant before they can leave. If the lab is not completed within the three-hour time frame students have until the next lab to demo and turn in their assignments. Homework assignments are available for evaluation but will not be used at this time. Comments were extracted from the lab files, labeled and organized into a .csv file. The first column is the label which is a 1 or a 0 to indicate if a comment is *Sufficient* or *Insufficient*. The second column is the raw comment text. For training the model all comments from both traditional and writing to learn sections have been combined into a single file. In *Figure 1* we can see a sample of the raw data before text normalization.

2.1 Text Normalization

We can normalize the text through a serious of processes such as tokenization, lemmatization, stemming, sentence segmentation and developing an edit distance metric. These steps allow us to

put our data into a consistent format making it easier to work with [5]. The first step in normalizing the comments corpus is to convert all words to lowercase and to remove all punctuation, then we can remove any unnecessary words using Stopword filtration.

2.2 Stopword Filtering

Stopword filtration allows us to remove certain words from our comment samples that occur frequently within the English language but add no value to the meaning of the text. This list includes words such as ‘a’, ‘the’, ‘an’, ‘but’, etc. For this paper, we will be using the list of stop words that are included with the NLTK library [4]. We can import common stopwords from nltk.corpus. A sample of the words included can be seen in *Figure 2*.

```
{'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there',
'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own',
'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of',
'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from',
'him', 'each', 'the', 'themselves', 'until', 'below', 'are', 'we', 'these',
'your', 'his', 'through', 'don', 'nor', 'me', 'were', 'her', 'more',
'himself', 'this', 'down', 'should', 'our', 'their', 'while', 'above',
'both', 'up', 'to', 'ours', 'had', 'she', 'all', 'no', 'when', 'at', 'any',
'before', 'them', 'same', 'and', 'been', 'have', 'in', 'will', 'on', 'does',
'yourselves', 'then', 'that', 'because', 'what', 'over', 'why', 'so', 'can',
'did', 'not', 'now', 'under', 'he', 'you', 'herself', 'has', 'just', 'where',
'too', 'only', 'myself', 'which', 'those', 'i', 'after', 'few', 'whom', 't',
'being', 'if', 'theirs', 'my', 'against', 'a', 'by', 'doing', 'it', 'how',
'further', 'was', 'here', 'than'}
```

Figure 2: NTLK Stopwords

To demonstrate how stopword filtering will affect each comment we can look at the before and after effects on a sample sentence. This shows the remaining words after all punctuation and stopwords have been removed.

Before Stop words Filtration

```
'Hello, this is an example message that we will use to demonstrate preprocessing.'
```

After Stop words Filtration

```
['Hello', 'example', 'message', 'use', 'demonstrate', 'preprocessing']
```

2.3 Tokenization

Tokenization is the process of splitting up a running body of text or text sample into individual words or ‘tokens’ much like the process used for tokenizing when building a parser for a programming language. This is necessary so that we can generate word frequencies and our dictionary which contains a list of all the vocabulary that occurs within our corpus. We can see the results of tokenization and text normalization on our data in *Figure 3*. The tokenized version of the comments will be the data we use to generate the dictionary and word frequencies to be utilized in calculating the inverse document frequency [4].

	label	comment	raw_length	tokenized	token_len
0	0	Explain the code to the user	28	[Explain, code, user]	3
1	1	Import math for the functions you will use later	48	[Import, math, functions, use, later]	5
2	1	Ask for the length of the tank, in meters	41	[Ask, length, tank, meters]	4
3	1	Ask for the diameter of the water tank, in meters	49	[Ask, diameter, water, tank, meters]	5
4	1	Ask for the number of residents the tank will ...	51	[Ask, number, residents, tank, serve]	5
5	1	Each person uses a constant 2 liters of water ...	53	[person, uses, constant, 2, liters, water, per...	8
6	0	Start the table.	16	[Start, table]	2
7	1	Divide the diameter by 2 to find the radius	43	[Divide, diameter, 2, find, radius]	5
8	1	Calculate the volume when no water has been us...	69	[Calculate, volume, water, used, height, max]	6
9	1	Calculate the days remaining before the water ...	54	[Calculate, days, remaining, water, runs]	5

Figure 3: Student Comments Corpus after Text Processing

2.4 Lemmatization

Lemmatization allows us to group sets of word inflections in our text by their root so that they are stored and analyzed as a single item in the dictionary. This allows us to store words such as 'am', 'are', 'is', and 'was' as a single word 'be' and it also removes pluralized forms of a word. [5]. For this paper, we implemented the text processing algorithm with and without lemmatization using the NLTK WordNetLemmatizer [4]. Without lemmatization, there were 647 unique vocabulary words and after lemmatization, there were 593 used to determine word frequencies. Despite the reduction in the vocabulary size this process reduced the accuracy of the Multinomial Naive Bayes classifier by 5% but improved the accuracy of Random Forest Classifier by 6%.

3. Data Exploration

The second phase of the project focuses on exploring some of the properties of the data and looking at some elementary statistics across the raw comment length and the tokenized length of comments. The first thing we will look at is the raw comment length of both sufficient and insufficient comments. We can see the raw frequency distribution of the comment lengths by characters in *Figure 4* and a statistical summary in *Table 1*. Here we can see that for an insufficient comment the average character length of a comment is 19.41 characters compared to an average of 52.66 characters for a sufficient comment. 75% of the comments are 60 characters long or less with a single outlier comment that is 382 characters.

Next, we will look at the Tokenized comment length. A 'token' is a single word within a tokenized comment after all punctuation and stop words have been removed. A statistical summary of token length can be seen in *Table 2* and the frequency distribution of the length of code comments can be seen in *Figure 5*. In *Figure 6* and *Figure 7* we can see the distribution of

the lengths when they are grouped by label where 0 indicates an insufficient comment and a 1 indicates a sufficient comment.

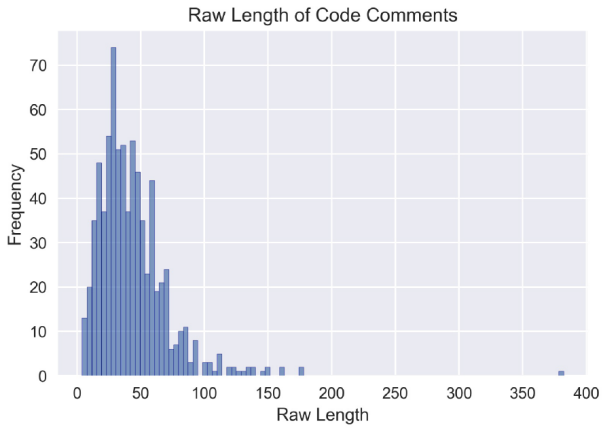

Figure 4: Frequency Distribution of Raw Comment Length

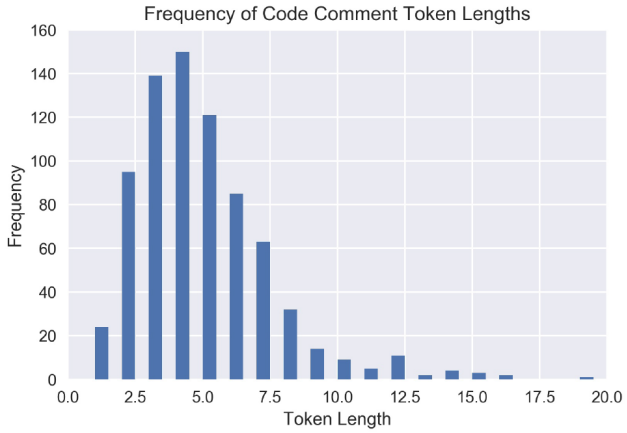

Figure 5: Frequency Distribution of Tokenized Comment Length

	Raw Comment Length						
Label	***Count***	***Mean***	***Std***	***Min***	***50%***	***75%***	***max***
0	210.0	19.41	9.08	4.0	19.0	23.75	69.0
1	551.0	52.66	28.28	199.0	46.0	60.0	382.0

Table 1: Statistical Summary of Raw Comment Length

	Tokenized Comment Length						
Label	***Count***	***Mean***	***Std***	***Min***	***50%***	***75%***	***max***
0	210.0	2.41	0.899	1.0	2.0	3.0	7.0
1	551.0	5.71	2.61	3.0	5.0	7.0	31.0

Table 2: Statistical Summary of Tokenized Comment Length

Here we can see that a clear difference between a sufficient comment and an insufficient comment is the token length where the average insufficient comment is 2.41 tokens and 75% are three or fewer tokens where the minimum length of a sufficient comment is three tokens and 50% of them are 5 tokens or more. When a comment is approximately 3 tokens in length this is where classification becomes difficult. A comment that is three words in length before tokenization is automatically classified as insufficient because it is too short to be considered significant in most cases.

Typically, most comments that are three words or less are restatements of the variable names or organizational markers within the code and can be considered to be trivial as they do not contribute to a deeper understanding of the code. However, some comments that were originally longer than three words end up with only three tokens after text processing but can sometimes be considered a sufficient comment candidate. We will see later that a majority of classification errors occur when a comment has a token length of three.

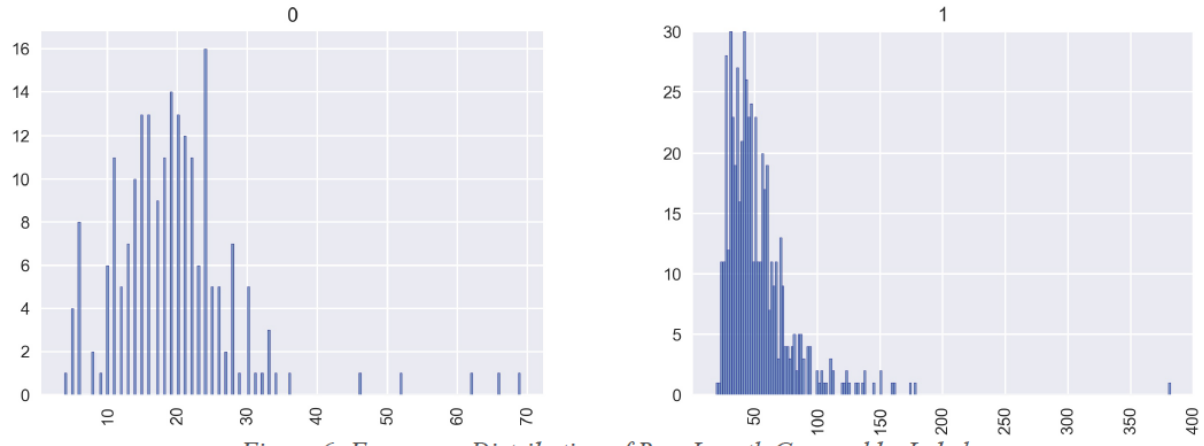

Figure 6: Frequency Distribution of Raw Length Grouped by Label

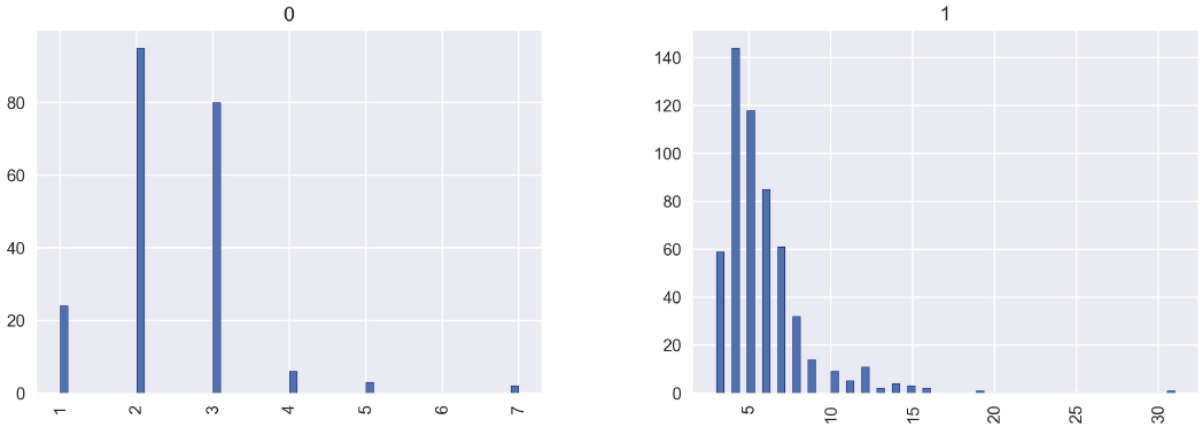

Figure 7: Frequency distribution of Token Length Grouped by Label

4. Comment Classification

The final phase consists of using our tokenized comments to create our weighted "Bag of Words" Model to train a Multinomial Naive Bayes Classifier and a Random Forest Classifier.

4.1 Bag of Words Model

In the "Bag of Words" model, we can represent the code comments corpus as an "unordered set of words" where the position of the word is ignored and we only maintain a record of the frequency of those words. Using the Count-Vectorizer from sci-kit learn that leverages our text processing function we can create a sparse matrix that will represent our comments as a 2D matrix of token counts where each row is for a single word and each column represents a comment [5]. This process generates a vocabulary of 647 unique words when our text processor does not include lemmatization.

We can look at how a single comment is represented within the matrix, for example, the fourth comment (at index 3) is "Ask for the diameter of the water tank, in meters" and the resulting bag of words for this comment is represented in *Figure 8*. Each token in our sentence is represented as an index and a frequency, for example, the word represented at index 282 in the sparse matrix is the word 'diameter' from our original sentence and it only occurs once in this comment. The properties of the resulting sparse matrix using our code comments corpus can be seen in *Table 3*.

```
(0, 20)       1
(0, 282)      1
(0, 447)      1
(0, 598)      1
(0, 638)      1
```

Figure 8: Bag of Words representation of Comment 4

Bag of Words Matrix Properties	
Shape	***(761, 647)***
Nonzero Occurrences	3569
Sparsity	0.725

Table 3: Code Comments Sparse Matrix

4.2 Inverse Document Frequency

The Term-Frequency – Inverse Document Frequency (TF-IDF) metric allows us to improve upon our Bag of Words model by adjusting the word counts based on their frequency in the corpus. The term frequency represents the importance of a word in a comment and the inverse document frequency is how important a word is in relation to the whole corpus. The TF-IDF value acts as a weight for a particular token where the most common tokens get the lowest weights. The resulting TF-IDF formula is represented by the equation in *Figure 9* where *N* is the number of total comments, *'tf'* is the term frequency and '*df'* is the number of comments containing *i*. [5]. Using the same comment as before, comment 4 at index 3, we can now see in *Figure 10* that each word has a weighted frequency attached to it.

$$w_{ij} = tf_{ij} \cdot log(\frac{N}{df_i})$$

Figure 9: Term-Frequency - Inverse Document Frequency

```
(0, 638)      0.326182315716
(0, 598)      0.367593481938
(0, 447)      0.537412483168
(0, 282)      0.489429700599
(0, 20)       0.479715040475
```

Figure 10: Bag of Words representation with weighted frequencies using TF-IDF

4.3 Training the Classifiers

To train the classifiers we created a pipeline using a Sci-kit Learn pipeline which allows one to pass in the bag of words, the TF-IDF frequencies and the model to be trained. 70% of the comments corpus is used for training and 30% is used for testing. The first tests exclude lemmatization and the second training tests include lemmatization. Interestingly enough lemmatization resulted in better outcomes for the Random Forest Classifier but worse on average for the Multinomial Naive Bayes Classifier.

Multinomial Naive Bayes Classifier				
	precision	*recall*	*f1-score*	*support*
0	0.88	0.26	0.41	57
1	0.80	0.99	0.89	172
avg/total	0.82	0.81	0.77	229

Table 4: Classification report Multinomial Naive Bayes without lemmatization

Multinomial Naive Bayes Classifier				
	precision	*recall*	*f1-score*	*support*
0	0.70	0.27	0.39	59
1	0.79	0.96	0.87	170
avg/total	0.77	0.78	0.74	229

Table 6: Classification report Multinomial Naive Bayes using lemmatization

Random Forest Classifier				
	precision	*recall*	*f1-score*	*support*
0	0.72	0.79	0.75	73
1	0.90	0.85	0.88	156
avg/total	0.84	0.83	0.84	229

Table 5: Classification report Random Forest without lemmatization

Random Forest Classifier				
	precision	*recall*	*f1-score*	*support*
0	0.79	0.84	0.81	62
1	0.94	0.92	0.93	167
avg/total	0.90	0.90	0.90	229

Table 7: Classification report Random Forest using lemmatization

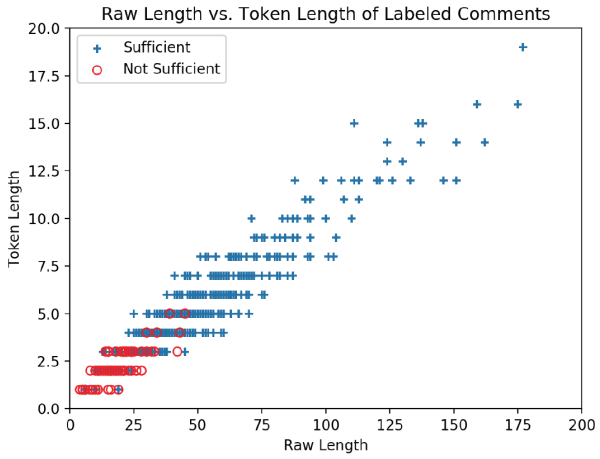

Figure 11: Predicted labels plotted by Raw length vs. Token length.

Overall, the Random Forest outperformed the Naive Bayes classifier in every training session. As a final illustration, we have a plot that demonstrates the predicted labels of our comment data using the Multinomial Naive Bayes Classifier. I used the Raw Length and Token Length to illustrate the relationship between the classification and the length of the comment. Here we can see that classification become mixed when the token length is around three which plays a strong role in the current error rate of our classifiers.

5. Future Research

As part of our current and future research, we continue to develop features for the classification of the student's Thinking Processes and Visual Organization [6]. Our aim is to develop a set of models that can be utilized to provide real-time classification and feedback to the student as they are programming. Future developments include classifying *sufficient* comments as *conceptual*, *literal*, *reflective* or *organizational* by utilizing convolutional neural networks and fine-tuning the classification of visual organization through the development of additional features to allow for the classification of sub-organizational units s as opposed to classifying the full document under a single strategy which becomes more useful as source code becomes more complex.

6. Conclusion

In this paper, we have discussed how to clean and process raw data using stop word filtration, tokenization, and lemmatization. We explored the data to discover some of its properties and to illustrate various characteristics of an *insufficient* comment versus a *sufficient* comment classification. Finally, we trained two supervised machine learning classifiers, the Multinomial Naive Bayes classifier and a Random Forest Classifier using the bag of words model and TFIDF weighting. We were able to achieve a precision of 82% using the Multinomial Naive Bayes classifier and using lemmatization a 90% precision rate on the Random Forest Classifier. Additional work includes experimenting with alternative supervised learning classifiers, comparing and contrasting traditional versus writing to learn comments, and creating a second level of classification for *sufficient* comments that classify them into literal, conceptual, reflective and organization.

Acknowledgement
This material is based upon work supported by the National Science Foundation under Grant No. DUE-1612132. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Mohammadi-Aragh et al., 2017] Jean Mohammadi-Aragh et al. “Coding the Coders: A Qualitative Investigation of Student’s Commenting Patterns”, In Proceedings of the 2018 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '18). ACM, Baltimore, MD, USA.
[2] Mohammadi-Aragh et al., 2017] Jean Mohammadi-Aragh et al. “Coding the Coders: A Qualitative Investigation of Student’s Commenting Patterns”, In Proceedings of the 2018 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '18). ACM, Baltimore, MD, USA.
[3] Mohammadi-Aragh et al., 2017] Jean Mohammadi-Aragh et al. “Coding the Coders: A Qualitative Investigation of Student’s Commenting Patterns”, In Proceedings of the 2018 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '18). ACM, Baltimore, MD, USA.
[4] Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O’Reilly Media Inc
[5] Dan Jurafsky and James H. Martin. Speech and Language Processing. 3rd ed. 2017. Internet: https://web.stanford.edu/~jurafsky/slp3/, [Accessed: Dec 6, 2017]
[6] Mohammadi-Aragh et al., 2017] Jean Mohammadi-Aragh et al. “Coding the Coders: A Qualitative Investigation of Student’s Commenting Patterns”, In Proceedings of the 2018 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '18). ACM, Baltimore, MD, USA.