Towards Semantic Classification: An Experimental Study on Automated Understanding of the Meaning of Verbal Utterances

Gnaneswar Villuri, Alex Doboli, Himavanth Reddy Pallapu

***Abstract*—Computationally understanding the meaning of verbal discussions in groups can improve group effectiveness by optimizing their interactions and allocating the needed physical and Cyber resources. Still, it is unknown to what degree the existing Machine Learning (ML) methods can automatically detect the type of verbal utterances, as a preliminary step towards automated meaning understanding. This paper presents a comprehensive experimental study of the performance of the main ML methods in classifying verbal utterances depending on their role during solving programming exercises. A model for interpretable classification using decision trees is also offered. The paper summarizes a set of requirements that new semantic classifiers must satisfy, as current ML methods are likely insufficient for the task. These requirements were experimentally validated.**

I. Introduction

Numerous modern applications in fields, such as smart homes, healthcare, corporate workplaces, digital classrooms, and more, consider human activities in social settings, including small groups addressing a joint task [1]. These applications often use speech (audio) signals to collect data, as verbal utterances are essential in human interactions. The main challenge here is to understand the meaning of the verbal utterances and the connection to the behavior and outcomes of a group, so that the group's activities can be optimized by managing group interactions and supplying the necessary physical and Cyber resources. As part of automated understanding, semantic classification distributes utterances into predefined categories based on the meaning of the utterances, not the static features (e.g., words) of the utterances.

Computationally understanding the meaning (semantics) of digital information has been studied in Artificial Intelligence (AI) [5]. Related work on computational semantics focuses on the automated identification of concepts in a corpus, like finding objects in images in semantic image segmentation [6], semantic grouping of design elements [7], creating dialogues [8], and generating outputs similar to humans [9]. Spoken Language Understanding (SLU), a main field in processing and interpreting human speech [2], [3], aims to extract meaning from spoken utterances, enabling machines to comprehend and respond to human speech in a manner similar to humans [4]. Current Machine Learning (ML) methods employ a variety of statistical concepts and metrics computed on the data features, including words, word sets, and word embedding [10], [11]. However, it is likely that they are insufficient for semantic processing, as meaning can be only partially related to statistical features [12]. For example, the statement "`Add the first three numbers in variable sum`" represents a solution idea, while the statement "`I agree, let's add the first three numbers`" is an analysis, even though it mostly uses the same words as the first sentence. The difference is produced by the meaning of the words "`I agree`", and not their frequency and similarity in the data corpus. Still, transformer models, like BERT [10], have shown a lot of promise in Natural Language Processing (NLP). Hence, it is still unknown what performance ML methods can offer in computationally understanding the meaning of speech utterances during problem solving, and what aspects reduce their effectiveness in automated understanding. Besides, the degree to which ML methods produce *semantic interpretability* of utterances similar to human understanding is unknown too.

This paper presents a comprehensive experimental study devised to characterize the degree to which current ML classification methods can distinguish verbal sentences into five categories depending on their semantics. The five categories are the activities conducted during team problem-solving for computer program development and are enumerated in Section II. Fifteen ML algorithms (shown in Table I) were studied, among which

This work relates to Department of Navy award N00014-22-1-2001 issued by Office of Naval Research.

are three Decision Tree Classifiers (DTCs). The three were then extensively analyzed, as compared to the other ML methods, Decision Trees (DTs) offer interpretable results. Interpretation insight is critical in finding the sentence semantics. The discussion of the experiments enumerates seven main characteristics of DTCs, and then a theoretical model is proposed for the DTC behavior as a way to understand their functioning for semantic verbal utterance classification. The DTC behavior model suggests three important characteristics, which we think, that any sentence semantics classifier should include. A comprehensive experimental study validates these characteristics. As the performance of current ML is insufficient, novel methods must be devised for the semantic classification of verbal utterances. We think that the present work contributes to the development of such new ML methods.

The paper is structured as follows. Section II analyzes the performance of semantic classification using ML methods. Section III models the DTC behavior. Section IV experimentally explores the DTC space. Conclusions end the paper.

II. Problem Description

Our goal is to design a system that automatically assigns a spoken utterance (sentence) or group of sentences into one of the following five categories, each category being labeled with a different color to simplify the presentation: **(i)** discussing problem requirements (color yellow), **(ii)** analyzing solution ideas (color green), **(iii)** describing high-level solution ideas (color grey), **(iv)** elaborating the high-level ideas (color blue), and (v) modifying an existing solution (color red). The ideas expressed during team problem solving pertain to the five categories, and understanding the idea types dynamics during the solving process is critical to improve team efficiency [13].

Examples: We illustrated next the five above categories. **(i)** A sample expressing problem requirements (labeled as color yellow) is shown by the next two sentences: "`Do we just find it every single combination? Because you want to find the sum that's closest to target.`" **(ii)** The next example was classified as analyzing solution ideas (labeled as color green): "`So, it says of length n, and I don't know if n is taken from the user or not, which doesn't specify. Should we make it taken from the user?`". **(iii)** The following sentence presents a high-level idea to solve the exercise (labeled as color grey): "`Have all combinations of numbers and add them.`" **(iv)** The next paragraph describes an elaboration (detailing) of a high-level solution idea (labeled as color blue): "`Hold the first two numbers constant to these two. We could do like a for loop. And then we could do a variable J. From index one to three.` **(v)** An example of text denoted as a modification of an existing solution idea (labeled as color red) is as follows: "`Wouldn't we need a saved sum? Also in the else if. Because we need to output the sum that's closest to the target. So in this if statement too, shouldn't we put, like, final sum or something like that, Yeah, final sum.`"

Classifying sentences based on their meaning (semantic classification) requires combining the meaning of words (i.e. nouns, pronouns), their relationships (e.g., verbs), and their features (like attributes) depending on the syntax and context of a sentence not only finding the co-occurrence of words interpreted as labels, as arguably in traditional ML methods. Also, classifying the verbal utterances of the problem solving dialog is different than text classification, traditionally studied in ML [7], [9], as utterances are shorter, more ambiguous, and often include grammatical errors.

A. Evaluation of Existing Classification Algorithms

The experimental evaluation used a comprehensive set of off-the-shelf classification algorithms to classify the ideas expressed during team problem-solving into the five categories above. Table I summarizes the results. Each row represents a different classifier. The considered algorithms were as follows: Decision Tree Classifier (DTC) [14], GOSDT DTC (GOSDT) [15], a mathematically optimal DTC for a given number of leaf nodes, the DTC in row 1 with its number of leaves constrained to be the same as the number of leaves in GOSDT (CONST.DTC), Logistic Regression Classifier (LR) [16], Random Forest Classifier (RFC) [17], K-Nearest Neighbor Classifier (KNN) [18], Support Vector Machine Classifier (SVM) [19], Naive Bayes Classifier (NB) [20], Linear Discriminant Analysis (LDA) [21], AdaBoost Classifier (AdaBoost) [22], Gradient Boosting Classifier (GBC) [23], Feedforward Neural Network (FNN) [24], BERT [10], DistilBERT [10], and RoBERTa [10]. Large Language Models (LLMs), like [25],[26], were studied too, but were not further analyzed as their performance for this task was low.

TABLE I: Performance of traditional classification algorithms

		Performance Metrics															
		Accuracy (%)				Precision (%)				Recall (%)				F1-score (%)			
		Min	Avg.	Max	Std. dev.	Min	Avg.	Max	Std. dev.	Min	Avg.	Max	Std. dev.	Min	Avg.	Max	Std. dev.
1	DTC	42	47	54	4	42	49	57	5	42	47	54	4	42	48	55	4
2	GOSDT	32	41	53	6	10	17	28	5	32	41	53	6	16	24	37	6
3	CONST.DTC	44	58	67	8	34	54	64	10	44	58	67	8	36	54	64	9
4	LR	42	48	60	6	56	63	69	4	42	48	60	6	34	38	51	6
5	RFC	49	57	62	4	49	61	70	7	49	57	62	4	46	52	58	4
6	KNN	51	59	68	5	50	57	65	5	51	59	68	5	51	57	66	6
7	SVM	38	47	63	8	58	65	72	5	38	47	63	8	30	39	58	9
8	NB	36	45	55	6	37	55	63	8	36	45	55	6	27	37	49	7
9	LDA	33	41	46	4	38	55	64	9	33	41	46	4	35	46	52	6
10	AdaBoost	41	54	62	8	38	53	62	8	41	54	62	8	36	51	60	9
11	GBC	48	56	62	4	46	60	64	6	48	56	62	4	43	53	59	5
12	FFNN	58	66	71	4	47	61	68	8	58	66	71	4	52	63	69	6
13	BERT	62	73	82	6	65	77	83	6	62	73	82	6	60	73	81	7
14	DistilBERT	68	74	83	5	60	72	81	7	68	74	83	5	63	73	82	6
15	RoBERTa	69	74	83	5	62	72	82	6	69	74	83	5	65	72	82	6

Experiments were conducted using the verbal discussions of 30 teams of undergraduate students (i.e. 30 datasets). Each team individually worked for twenty minutes to solve a programming exercise. Their verbal discussions were utilized for speaker tracking and then converted into text [27]. The individual dataset sizes were between 56 and 289 sentences. In total, 3714 sentences were used for training and testing the classifiers. 105 classification runs were executed as part of the experimental evaluation.

Table I summarizes the performance of the fifteen algorithms for classifying the discussion sentences into the five categories. Each row in the table is for another classifier. The measured performance metrics were accuracy, precision, recall, and F1-score [11]. The table columns describe the minimum (Min), average (Avg.), maximum (Max), and standard deviation (std.dev.) for each of the four metrics. The sensitivity and specificity of the algorithms were also found but were not included in the table due to space constraints.

The best results were obtained for the three transformer-based deep learning models, BERT, DistilBERT, and RoBERTa. Their average accuracy is around 74%, their maximum accuracy about 83%, and their minimum accuracy between 62-69%. The low standard deviation shows that most accuracy values were close to the averages, e.g., 15 values out of 21. Next, classifiers LR, RFC, SVM, and GBC offered an average accuracy between 48%-57%. The third cluster included the classifiers KNN, NB, LDA, AdaBoost, NB, and FNN with an accuracy between 41%-66%. Their standard deviations were also small, in the range 4%-8%, suggesting that most datasets produced an accuracy close to the average values. Accuracy and recall were the same, as the datasets were highly imbalanced, i.e. 91% of the sentences belonged to two out of the five categories. Hence, the True Negatives and False Positives were zero. The F1-score supports the previous observations, however, its low values suggest that accurately classifying verbal utterances into the five categories is difficult. Similar trends were observed for sensitivity and specificity too.

The three DTC classifiers, e.g., DTC, GOSDT, and CONST.DTC offer an average accuracy between 41%-58%, with the lowest accuracy between 32% and 44%, and the highest in the range 53%-67%. The number of leaf nodes, hence produced categories, was between 18 and 102 for DTC, suggesting overfitting of some of the classifiers. In contrast, GOSDT constrains its number of leaves [15], hence, its DTs had between 3 and 12 leaves. The standard deviation was small, with 69-73 instances out of 182 having an accuracy close to the average value.

The next subsection focuses on the detailed discussion of the DTC and GOSDT results. The two offer result interpretability (explainability), like which words decide the classification results, similar to human understanding. We also studied the interpretability of the three transformer models using Captum toolkit, a "transformer-interpret" Python module that calculates metrics, e.g., prediction score, attribution score, and word importance. However, we argue that score-based interpretation is weaker than the split node conditions of DTCs, as they offer probabilistic descriptions with significant overlapping between how different word clusters decide classification. For example, the following sentence "`You add two numbers and then the numbers that are closest to zero in the original array`", labeled as blue by human raters, was analyzed with Captum toolkit to understand how

the words decided the category of the sentence. The analysis produced ambiguous insight, as the words "you", "that number", and "array" suggest colors blue and green, while "then the numbers that are closest to zero in the" point towards colors grey and yellow. Hence, finding distinguishing word structures, like in DTCs, is unlikely. Finally, the flat description offered by metrics, as opposed to the hierarchical nature of the split nodes in DTCs, makes interpretation harder too.

B. Discussions of the Results

This section focuses on the analysis of the main miss-classification results produced by DTCs, in particular the GOSDT classifier. GOSDT DTC is arguably one of the most powerful classification methods, as it produces decision trees (DTs) with minimum error using a branch-and-bound method guided by bounding rules that are mathematically proven to produce optimal results [15].

1. The majority of the sentences were classified through exclusion. Classification is traditionally defined as using the distinguishing features of data to separate between categories so that the items of a category share a set of common features [11]. However, the experiments showed that the majority of the sentences were classified through exclusion, e.g., their words were not among the words used in the split node conditions. For example, from a dataset of 607 sentences, 419 sentences were classified through exclusion, e.g., they did not include the words mentioned in the split conditions. The leaf nodes selected through the presence of two or three words classified only a small number of sentences, i.e., 12 sentences out of a total of 607, but the produced classification accuracy for these nodes was high, 91%.

2. Distinguishing words were not used in classification. We manually identified the words that we considered to suggest a certain category. For example, words that indicate specific programming tasks, like the word `increment`, are more likely to suggest elaboration steps (color blue). However, even though they are important in deciding the category, such words were often not selected to form the conditions of the split nodes. For example, the word `main`, which is important in C programming and used in every program, was never part of the classification conditions of the DTCs. Similarly, words suggesting high-level solutions (color grey) were never used in the associated classification conditions either. This means that there is a gap between the words used by the DTC classifier and the words used by humans to interpret a sentence as being in one of the five categories.

3. Limitations of frequency-based classification. Traditional classification methods, including DTC, use the frequency of words to select the words that are part of the split conditions [11]. More frequent words are likely to be used, such as the words `it`, `that`, and `we` were used in the split node conditions, even though they do not offer much help in interpretability. Less frequent words were ignored when setting up the classifier conditions, in spite of them having a high role in explaining the category selected by humans for a sentence.

4. Categories were overlapping. The words selected for creating the split node conditions can appear in sentences pertaining to multiple categories. For example, the word `equals` misclassified some sentences as analysis (color green) instead of solution elaboration (color blue). These cases support the previous observation, e.g., the good DTC classification words are not the same as the good words for human interpretation.

5. Stop words improved classification accuracy. Keeping stop words, like `the`, `that`, and `and` improved classification accuracy. This result conflicts with the usual procedure of eliminating stop words during NLP.

6. Agreement words were not picked up. Agreement and disagreement words, like `OK`, `yeah`, `no`, and so on, are decisive in indicating an analysis situation, e.g., when someone agrees with a previous statement. For example, the statement `Yes, let's sum up values` indicates an agreement with a previous statement, like the sentence `We will add up the value in a variable`, which is a description of a high-level solution (color grey). However, the first statement suggests the analysis of this idea by another participant (color green). The decisive word, i.e. the word `Yes`, is not used by the DTC in its split node conditions to classify these sentences.

In conclusion, the experiments show that even though DTCs are expected to offer interpretable ML, the words selected in classification are not the words that humans use in understanding the nature of the sentences. This observation can be justified by the fact that the nature of the metrics used in decision tree induction [15], [11] are not linked to the way humans understand natural language semantics. As expected, the classifiers learned the type distributions of the sentences, but the classification results were poor when the type distributions of the test data were different. Hence, it is reasonable to conclude that the classifiers did not learn the semantics of the sentences with respect to their kind.

The next section discusses the modeling of the DTC behavior, as a step towards designing new methods to

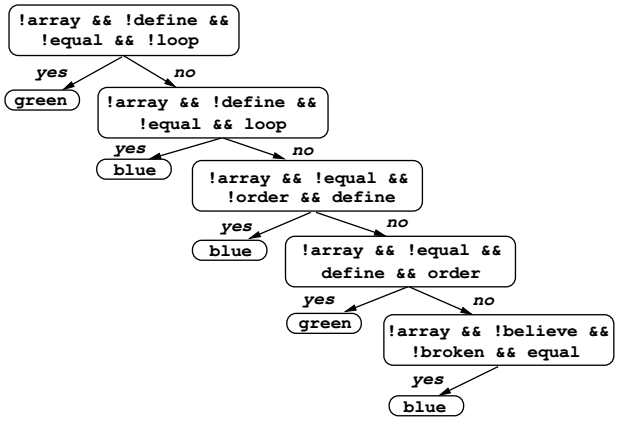

Fig. 1: Example of decision tree classifier produced for a statement sample.

computationally find the meaning of sentences.

III. Decision Tree Classification Modeling

Figure 1 illustrates a decision tree classifier (DTC) produced using the GOSDT method for one dataset including the statements produced by one team during problem-solving. The accuracy of the classifier was 57%. Similar classifiers were produced for the other datasets too.

The behavior of the DTCs can be modeled as follows:

1. Split node conditions. The conditions of the split nodes in a decision tree (DT) satisfy the following constraints:

$$SC_p = \prod_{i=1}^{k} !word_i \prod_{j=1}^{n-k} word_j \quad (1)$$

Variable SC_p represents the condition at the split node p, variable $!word_i$ indicates that word $word_i$ is not present in a sentence, and word $word_j$ describes that word $word_j$ is present. The total number of words used to build condition SC_p is n. In a DTC, all split node conditions use the same number of words, n.

Observation: Any word first occurs in a negated form in the condition of a split node before it possibly occurs in a non-negated form in the condition of a split node that has the former node as an ancestor.

As shown in Figure 1, the top split node in any DT has a condition stating that a set of words should not occur in a statement, i.e. words `array`, `define`, `equal`, and `loop`. This supports the observations that DTs mostly classify through exclusion, as the root nodes classify most of the sentences. The conditions of the following split nodes are similar to the conditions of their parents because one, at most two variables were changed. For example, the second split node condition states that word `loop` should be present, while the rest of the condition is the same as the condition of the root. As compared to the second split node, the condition of the third split node differs through the requirement that word `define` is present and word `order` not present (hence, two different words). Also, we observed that the conditions of the split nodes do not require more than two words to be present.

Theorem: The decision making at split nodes starting from the second descendant of the root depends only on the words involved in the condition of the current node, and not on any word present in the conditions of the split nodes from the root to the current node.

Proof: As soon as there is one word that occurs in a negated and non-negated form in the conditions, let's call that word a, the equivalent equation expressed by the previous split nodes is $a \vee !a$ ..., which is always true, and hence does not influence the conditions of the current split node. For example, in Figure 1, word `loop` should not be present for the root but present for its child, hence the classifications of the split nodes starting from the second level downwards depend only on their conditions.

A consequence of this theorem is the fact that a DT separately models a group of sentences, e.g., the sentences classified on the branch *yes* of the split nodes, without having the possibility to represent any context set-up by the preceding sentences. However, the context can be important in deciding the meaning of a sentence (labeled through a certain color).

2. Relationship between the number of leaves and the number of words used in the conditions. The number of DT leaves relates to the number of words in the conditions of the split nodes. This is a consequence of equation (1) and the previous theorem. The next theorem captures this aspect.

Theorem: The total number of words used in the split node conditions satisfies the following bound:

$$Number_{words} \leq Number_{leaves} \times Number_{variables} \quad (2)$$

where $Number_{words}$ is the total number of words used in the conditions of a DT, $Number_{leaves}$ is the total number of leaves of the classifier, and $Number_{variables}$ is the number of words used in the conditions of the split nodes.

Proof: The upper bound in equation (2) results directly from the observation that each condition must have at least one word that is common with the condition of an ancestor split node.

Relationship (2) is important in explaining the characteristics of the produced DTC, including the observa-

tions summarized in Section II.B. The theorem indicates that building a DTC can be restated as identifying the cardinality-constrained set of words among those used in conversations, so that using these words in creating conditions of the form in expression (1) allows associating a type (color) to the sentences, so that the error of this association is minimized. The cardinality of this set is constrained by the number of leaves of the resulting classifier, and this number relates to the generalization capability of the classifier, i.e. a large number of leaves overfits the classifier and reduces its accuracy for unseen sentences.

3. Overall behavior of a DTC to classify sentences. Sentence classification can be described as sentence partitioning into a constrained number of bins. Each bin is for a sentence type (labeled using a color) and is selected by conditions according to equation (1). Each bin is a leaf node in the DTC.

Theorem: The conditions used to select the majority of the nodes mainly include expressions $!word_j$ stating that $word_j$ is not part of the classified sentence. Hence, sentences are mostly classified through exclusion.

Proof: This theorem corresponds to the first experimental observation in Section II.B. Instead of describing a sentence through the words it includes, a sentence can be described by its complement with respect to all the words, e.g., the words it does not include from the set of all words. Given that the classified sentences have few common words, it is more likely that the words placed in a bin are described by the common words that they do not include rather than the common words that they include. Hence, finding the conditions (1) requires finding the optimized set of words not present in sentences.

Observation five in Section II.B is also a consequence of the above theorem. As conditions (1) must also include words present in the statements, and as stop words, like words `the`, `to`, and `that`, and pronouns, e.g., words `it`, `you`, and `we`, are among the most frequent words in the statements, eliminating them before classifier construction reduces the classifier accuracy. The selected words to be checked for inclusion are the more generic words, like `loop`, `define`, `order`, `know`, and `equal`, and less the programming-specific words, like words `array`, `variable`, `initialize`, `scan`, and so on, as expected. Also, modifying the classifier to check for words `equal` and `loop` being together in a split node condition decreases the accuracy of the classifier, as the ratio of sentences including both words is small. Instead, a human rater uses both words to increase his/her confidence in placing a statement into a certain category, like elaborating a solution. Hence, the theorem supports the second observation in Section II.B, as the words used to set-up the classifier are not the words used by humans to interpret, i.e. the words that guide the human understanding of a sentence.

Observations: The main limitations and opportunities to improve the DTC accuracy can be summarized as follows based on the modeling of DTC behavior:

1. Grouping words based on their meaning. Word grouping is justified by two facts: (i) the frequency of individual words is often low, and (ii) the number of words used in the split node conditions is constrained by the number of DTC leaf nodes to avoid overfitting. Moreover, the used words are sometimes synonyms or have similar meanings in the programming context, i.e. the words `read`, `scan`, and `enter`, or words `copy`, `set`, and `save`. Such words can be clustered together, so that the cluster is used in the split node conditions, not the individual words. Grouping can also increase the likelihood of less frequent yet important words to be pick up for classification.

2. Capturing the syntax of sentences. Syntax is essential in understanding the roles of individual words to communicate a certain message. However, all classification algorithms in Section II.A ignore syntax. Syntax supports expressing flexible and hierarchical structures, which further stresses the need to utilize syntactic features in classification.

3. Considering the sentence context. Each sentence is separately classified, even though the context is important in deciding the type of a sentence. For example, a participant might suggest "`All array values should be sorted`", which is a high-level solution idea (color grey). Another participant might then immediately say "`Yes, I agree. The values should be sorted`". This statement, even though similar to the first statement, is an analysis and agreement by the second participant, thus another category (color green). Without using the previous statement, the second statement is classified as an elaboration too (color blue).

Effectively addressing the three issues is beyond incrementally changing existing classification methods, as it requires significant rethinking of the classifiers for sentences. As a first step towards this goal, the next section discusses our attempts to experimentally study items 1, 2, and 3 by changing the use of GOSDT.

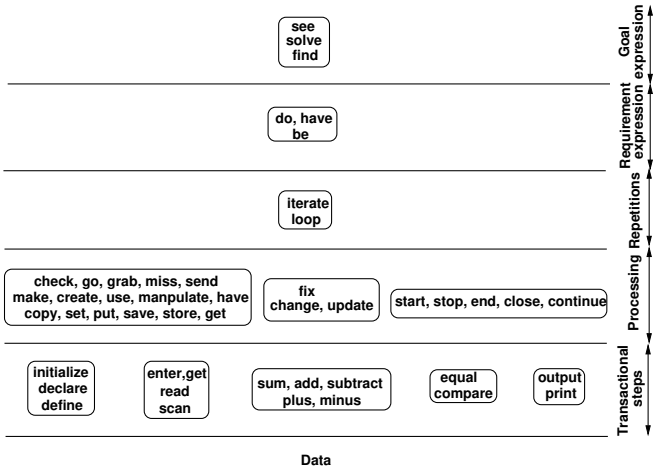

Fig. 2: Word clustering based on programming semantics.

IV. Experimentally Exploring the Space of Decision Tree Induction Algorithms

The first three observations ending Section III were used to extend GOSDT. Then, they were experimentally studied.

1. Grouping words based on their meaning. We manually clustered the words used in team conversations based on their similarity with respect to their meaning in programming. Figure 2 depicts a sample of the produced word clustering for verbs. The clusters were organized into the following levels depending on how the relations described by verbs relate to data: (A) **Transactional steps**: These verbs indicate precise actions on data. For example, the cluster including words `define`, `declare`, and `initialize` refers to creating the variables used in a program. The cluster of words `scan`, `read`, `enter`, and `get` describes the input of values. (B) **Processing**: These verbs describe higher-level processing, which involves several transactional steps. For example, the cluster formed by words `change`, `update`, and `fix` describes the modification of data expressed at a broader, less specific level. Some word clusters express temporal aspects, like the words `start`, `stop`, `end`, `close`, and `continue`. Verbs can refer to ambiguous, unknown, or aggregated elements, i.e. variables. (C) **Repetitions**: Verbs, like `loop` and `iterate`, present the repetition of transactional steps or processing. (D) **Requirement expression**: The verbs are used to express requirements for data, transactional steps, processing, and repetitions, like their required values and conditions. For example, the sentence "`The sum must be positive`" presents a required condition for processing. (E) **Goal expression**: Words, such as `find`, `solve`, and `see`, describe a goal that must be achieved during problem solving. For example, the sentence "`Find the largest value of the array`" indicates a goal suggested as part of the solution.

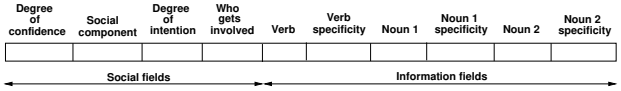

Fig. 3: Sentence encoding.

In addition to their programming semantics, verbs were also manually characterized with respect to their specificity, e.g., the degree to which they describe a precise relationship. For example, the verb `define` was considered to be more precise than the verb `declare`, as the word `define` is a keyword in C language, and the verb `initialize` is less specific than the verb `declare`, as it misses information on how initializing should be performed. Similarly to verbs, nouns were also manually clustered depending on their programming-related semantics, and their specificity was defined too. Each sentence was then encoded as shown by the last six fields of Figure 3: the verb of the sentence, the noun preceding the verb ($Noun$ 1), and the noun succeeding the verb ($Noun$ 2).

2. Capturing the sentence syntax. None of the traditional classification methods in Section II.A explicitly considers the sentence syntax, even though the syntax is essential in deciding its meaning. For example, the sentence "`You manipulate the numbers to get as close as you can to the target`" describes a high-level idea of the problem solution (color grey). But the sentence "`Numbers can be close to target as you manipulate them`", which uses the same words but with a different syntax, expresses analysis (color green).

The following incremental change of GOSDT was performed in order to consider some facets of syntax: three separate classifiers were built, one trained using the verbs in the sentences, the second based on the nouns preceding the verbs ($Nouns1$), and the third classifier using the nouns following the verbs ($Nouns2$). Then, the outputs of the three classifiers were utilized in two separate modes to predict the sentence type (color): (i) the winner-takes-it-all mode outputs the most frequent prediction of the three classifiers, and (ii) the biased mode uses the predictions of the verb and second noun, with the noun having a higher weight than the verb. The verb and the positioning of the two nouns give a rudimentary description of the syntax while keeping

GOSDT unchanged.

3. Considering the sentence context. As explained by the second theorem in Section III, capturing the context of a sentence is challenging, and requires significant changes to the GOSDT classifier. The capability to relate to multiple connected sentences must be added. Also, team dialog sentences can communicate social aspects in addition to the programming-related content. For example, in the sentence "`You have to set the difference to be zero`", the first verb (`have`) indicates an intention.

The encoding in Figure 3 captures the degree of intention and who gets involved in this description. The created encoding considers that verbs, like `have to`, `must`, and `need`, express stronger intentions than verbs `can`, `think`, and `hope`. Field *Who gets involved* indicates if the intention refers to the speaker (e.g., pronoun `I`), another present participant i.e. pronoun `you`, a participant not present (i.e. pronouns `he`, `she`), or groups (like pronouns `we`, `they`, and so on).

Similarly, the encoding for fields *Degree of confidence* and *Social components* in Figure 3 captures the degree of confidence of the speakers. Verbs, like `believe`, `guess`, `assume`, `say`, `consider`, `remind`, etc., indicate a certain level of confidence about the statement that follows the verbs. Field *Social components* refers to who is involved in the confidence stating, i.e. the speaker (pronoun `I`), or all present participants, such as the pronoun `we`.

A. Experimental Results

Table II shows the classification performance using the GOSDT algorithm extended as follows: Rows (1)-(3) (labeled as *Encoding-x*) describe the classifier trained using data encoded as shown in Figure 3. Rows (4)-(6) (labeled as *Enc-Noun2-x*) uses the nouns/pronouns following the verb in a sentence ($Noun2$) utilizing the same encoding, and Rows (7)-(9) (denoted as *POS-Nouns*) uses the same nouns/pronouns but without encoding. Rows (10)-(12) (named *POS-Maj.-x*) implements the winner-takes-it-all mode discussed in Subsection IV.2. Rows (13)-(15) (named *En-POS-Nouns-x*) use the same mechanism like rows (7)-(9) but with encoded nouns. Rows (16)-(18) *labeled* En-POS-Maj.-x use the same strategy as rows (10)-(12) but with encoded parts of speech (POS). Finally, rows (19)-(21) implement AdaBoost scheme (*AdaBoost-x*), in which individual classifiers were devised to classify large chunks of data identified by a single split node condition (like the large chunks selected by the branch `yes` in Figure 1).

Each classifier type was constructed for three increasingly larger training sets (labels x of the table rows): 390 sentences for label 1, 607 sentences for label 2, and 880 sentences for label 3. To offer a comprehensive study, the following classification cases were studied: (1) training on datasets 1 and 2, and testing on datasets 3-14, (2) training on datasets 1-3, and testing on datasets 4-14, and (3) training on datasets 1-6, and testing on datasets 7-14.

The experimental results in Table II show that the average accuracy is between 41%-48%, the minimum accuracy is between 25%-41%, and the maximum accuracy is between 53%-65%. The AdaBoost strategy, which usually performs well, was less effective in this experiment, with the average, minimum, and maximum accuracy being 45%, 28%, and 65%, respectively. The best results were obtained for encoded POS using the winner-takes-all option (line 18) with the average, minimum, and maximum accuracy being 48%, 41%, and 59%, respectively. The standard deviation was only 6%, which shows that the results were consistently obtained. Better results were produced also by using encoded nouns for training (line 15) with an average accuracy of 47%, a minimum accuracy of 33%, and a maximum accuracy of 58%. Also, using encoding and information about POS and syntax, like the nouns after verbs, improves classification.

The values obtained for the other metrics, like precision, recall, and F-1 score can be explained as follows. The used training and testing datasets were imbalanced: sentences labeled as green and blue were 80.5% in the training datasets, and 88% in the test sets. Experiments produced more false negatives than false positives because the classifiers also predicted colors other than green or blue and the number of sentences of colors other than green or blue was significantly less, i.e. 12% in some of the testing datasets. Hence, the precision values were greater than the recall values, as shown by 34 of 42 experimental situations, e.g., 81% times. F-1 metric offered the same conclusions. The F-1 value dropping reflecting the imbalanced datasets were similar to the results in Table I. Hence, encoding, and information about POS and syntax did not help in mitigating the effect of data imbalances.

The performance results can be explained as follows: Colors green and blue contributed to more than 80% of the total sentences in the dataset. Only one team, i.e., team 4, had the green and blue contributions at 75%. Since the two colors consistently labeled most instances, the accuracy obtained for the teams was mostly around

TABLE II: Performance of the extended GOSDT algorithm.

		Metrics															
		Accuracy				Precision				Recall				F1-score			
		Min	Avg.	Max	Std. dev.	Min	Avg.	Max	Std. dev.	Min	Avg.	Max	Std. dev.	Min Min	Avg. Avg.	Max Max	Std. dev.
1	Encoding-1	31	44	55	7	24	45	65	11	31	44	55	7	24	39	53	8
2	Encoding-2	37	44	61	7	31	45	63	8	37	44	61	7	34	42	60	8
3	Encoding-3	39	45	53	5	37	46	57	6	39	45	53	5	37	44	54	5
4	Enc-Noun2-1	30	44	60	9	25	43	75	14	30	44	60	9	20	34	52	10
5	Enc-Noun2-2	25	43	56	9	7	40	62	16	25	43	56	9	12	33	49	10
6	Enc-Noun2-3	33	45	61	9	30	46	64	11	33	45	61	9	25	37	55	10
7	POS-Nouns-1	28	44	57	9	13	45	74	20	28	44	57	9	16	31	46	9
8	POS-Nouns-2	32	46	59	8	26	47	70	13	32	46	59	8	23	35	51	9
9	POS-Nouns-3	30	45	55	9	30	56	75	14	30	45	55	9	18	31	43	9
10	POS-Maj.-1	32	45	58	9	13	47	74	19	32	45	58	9	19	33	49	9
11	POS-Maj.-2	33	46	59	9	19	49	72	15	33	46	59	9	26	36	51	8
12	POS-Maj.-3	32	45	58	11	31	54	74	15	32	46	58	11	23	35	49	10
13	En-POS-Nouns-1	25	43	57	9	13	44	65	15	25	43	57	9	15	34	51	10
14	En-POS-Nouns-2	35	45	61	7	30	45	64	8	35	45	61	7	29	39	51	6
15	En-POS-Nouns-3	33	47	58	8	39	49	60	7	33	47	58	8	26	42	54	9
16	En-POS-Maj.-1	35	44	55	6	29	45	60	9	35	44	55	6	27	39	54	7
17	En-POS-Maj.-2	28	41	53	8	33	42	58	8	28	41	53	8	29	41	55	8
18	En-POS-Maj.-3	41	48	59	6	37	47	61	7	41	48	59	6	39	47	59	7
19	AdaBoost-1	29	46	53	7	20	45	60	11	29	46	53	7	20	40	49	8
20	AdaBoost-2	28	45	53	7	28	46	55	8	28	45	53	7	23	41	51	8
21	AdaBoost-3	28	45	65	12	25	44	65	14	28	45	65	12	25	42	63	12

the mean contributing to a low standard deviation. But, if the sentences labeled as blue were significantly more than those denoted as green (e.g., more than 30%), then those teams consistently offered the least accuracy, like teams 9 and 12. For example, team 9 had 32% of its sentences as green and 64% as blue. Team 12 had 28% of the sentences as green and 62% as blue.

The following words are specific to the individual categories: discussing problem requirements (color yellow) has the individual words `array` and `chance`, and the words `answer`, `array`, and `it` appearing together. The category for elaborating high-level ideas (color blue) has the individual word `loop`, the set `are`, `it`, `right`, and the set `care`, `oh`. The results support the conclusion that the words used in the split node conditions of the classifiers are usually not the words interpreting a sentence assignment to a category.

As a reference, we also trained individual classifiers for each of the studied teams. The obtained accuracy was high. The best classifiers had an accuracy of 85.71% with 10 leaves, and with 33.93% of the data being classified by the top split node. The words used in the split node conditions were `dont`, `first`, `oh`, `so`, `add`, `time`, `care`, and `check`. Another strong classifier had an accuracy of 83.87% with 14 leaf nodes, and with 35.48% of the data being decided by the top split node. The words used in the conditions were `don't`, `it`, `oh`, `right`, `beginning`, `plus`, `three`, `thing`, and `are`. The lowest accuracy was 60.36% with 16 nodes and 57% of the data classified by the top split node, and 64.64% with 14 leaf nodes with 70% of all sentences classified by the top node. Hence, every customized classifier used a different set of words for its conditions, and the number of leaf nodes was weakly correlated to the classifier accuracy.

B. Discussion: Needs for Future Sentence Classifiers

The theoretical model for the DTC behavior and the subsequent experimental results suggest that the methods for semantic classification of verbal utterances must address the following aspects:

1) The good words for automated classification are often not same as the good words for human semantic interpretation. This difference suggests that the approach to classify spoken sentences must be rethought as compared to existing DTCs.
2) Classification conditions must emphasize the present words not the absent ones, like the conditions of the current DTCs do. Word clustering can help using rare yet important words, like words that decide the sentence category.
3) The sentence context is important in utterance classification, as the current DTC induction methods produce DTs in which every split node acts independent of the other split nodes. This requires extending the split conditions beyond the form in equation (1).
4) Information on the sentence syntax could be used by the DTCs, including the capacity of syntax to express hierarchical structures, and structures with

similar meaning but different POS positioning. We think that insight from linguistics is needed.

5) The DTC problem is a constrained optimization problem due to the overfitting constraints. This bounds the total number of words used in classification.

V. Conclusions

This paper presents a comprehensive experimental study on using fifteen existing ML methods to classify based on their semantics the verbal utterances of a team during code development into five categories. The categories represent the activities during programming. Three DTCs were considered, as interpretability of the classification results is important in semantic understanding. As none of the methods offered sufficiently high accuracy and interpretability, a model was devised to express the behavior of DTCs, so that the causes of their poor accuracy could be better understood. The model suggested that elements, like word encoding, syntax, and context can help to improve performance. Two additional experimental studies confirmed this insight. Future work will address the devising of novel semantic classifiers, that will incorporate the requirements identified by this work.

References

[1] A. Stefanini, et al., "Patient satisfaction in emergency department: Unveiling complex interactions by wearable sensors", Journ. Busin. Res., 129, pp. 600-611, 2021.

[2] Tur, G., & De Mori, R. (2011). Spoken language understanding: Systems for extracting semantic information from speech. John Wiley & Sons.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, Brian Kingsbury. *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*. IEEE Signal Processing Magazine, 29(6):82-97, 2012.

[4] Picovoice. (2022). SLU 101: Introduction to Spoken Language Understanding. Retrieved from https://picovoice.ai/blog/spoken-language-understanding-slu/

[5] D. Lenat and G. Marcus, "Getting from Generative AI to Trustworthy AI: What LLMs might learn from Cyc", *arXiv*, 2308.04445, 2023.

[6] Z. Yang, et al., "Fully Convolutional Network-Based Self-Supervised Learning for Semantic Segmentation", *IEEE Trans. Neural Networks*, 35(1), pp. 132–142, 2024.

[7] P. Duan, et al., "Towards Semantically-Aware UI Design Tools: Design, Implementation, and Evaluation of Semantic Grouping Guidelines". *Proc. Workshop AI and HCI*, 2023.

[8] R. Thoppilan, et al., "LaMDA: Language Models for Dialog Applications", *arXiv*, 2201.08239, 2022.

[9] A. Srivastava, et al., "Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models", *arXiv*, 2206.04615, 2023.

[10] https://huggingface.co/docs/transformers/model_doc/.

[11] J. Han, et al., "Data Mining. Concepts and Techniques", *Elsevier*, 2012.

[12] N. Chomsky et al., "The False Promise of ChatGPT", *The New York Times*, March 8, 2023.

[13] Removed because of blind review.

[14] https://scikit-learn.org/stable/modules/tree.html.

[15] J. Lin, et al., "Generalized and scalable optimal sparse decision trees", *Proc. International Conference on Machine Learning*, 571, 2020.

[16] https://scikit- learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[17] https://scikit- learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[18] https://scikit- learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[19] https://scikit-learn.org/stable/modules/svm.html.

[20] https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes.

[21] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

[22] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html.

[23] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

[24] https://github.com/fchollet/keras.

[25] Y. Chae, T. Davidson, "Large Language Models for Text Classification: From Zero-Shot Learning to Fine-Tuning". *SocArXiv*, https://10.31235/osf.io/sthwk, 2023.

[26] X. Sun, et al., "Text Classification via Large Language Models", *arXiv*, 2305.08377, 2023.

[27] Removed because of blind review.