Bot2Vec: Learning Representations of Chatbots

Jonathan Herzig[1] **Tommy Sandbank**[*,2] **Michal Shmueli-Scheuer**[1]
David Konopnicki[1] **John Richards**[1]

[1]IBM Research
[2]Similari

{hjon,shmueli,davidko}.il.ibm.com, tommy@similari.com, ajtr@us.ibm.com

Abstract

Chatbots (i.e., *bots*) are becoming widely used in multiple domains, along with supporting bot programming platforms. These platforms are equipped with novel testing tools aimed at improving the quality of individual chatbots. Doing so requires an understanding of what sort of bots are being built (captured by their underlying conversation graphs) and how well they perform (derived through analysis of conversation logs). In this paper, we propose a new model, BOT2VEC, that embeds bots to a compact representation based on their structure and usage logs. Then, we utilize BOT2VEC representations to improve the quality of two bot analysis tasks. Using conversation data and graphs of over than 90 bots, we show that BOT2VEC representations improve detection performance by more than 16% for both tasks.

1 Introduction

As conversational systems (i.e., chatbots) become more pervasive, careful analysis of their capabilities becomes important. Conversational systems are being used for a variety of support, service, and sales applications that were formerly handled by human agents. Thus, organizations deploying such systems must be able to understand bots behavior to improve their performance. In many cases, such an analysis can be viewed as a classification task whose goal is to check whether a bot or a particular instance of a conversation satisfies some property (e.g., is the conversation successful?). Models for these downstream classification tasks should benefit from conditioning on representations that capture global bot behavior.

For a conversation itself, there exists a natural way to represent it as the concatenation of the human and bot utterances. As for a bot, the question of its representation is more complicated: bots are complex objects that execute logic in order to drive conversations with users. How should they best be represented?

Many commercial companies provide bot programming platforms. These platforms provide tools and services to develop bots, monitor and improve their quality. Due to the increasing popularity of bots, thousands or tens of thousands of bots could be deployed by different companies on each platform[1]. Although bots might have different purposes and different underlying structures, the ability to understand bot behavior at a high level could inspire new tools and services benefitting all bots on the platform. In this work, we explore a commercial platform, and study different bot representations.

Inspired by the success of recently proposed learned embeddings for objects such as graphs (Narayanan et al., 2017), nodes (Grover and Leskovec, 2016), documents (Le and Mikolov, 2014) and words (Mikolov et al., 2013a), we propose a new model, BOT2VEC, that learns bot embeddings, and propose both content and graph based representations. While previous graph embedding representations consider static local structures in the graph (Narayanan et al., 2017; Grover and Leskovec, 2016), our graph representation is based on dynamic conversation paths. As bots are usually represented on bot platforms as some form of directed graph, with conversations represented as traversals on the graph, this approach seems reasonable. It captures the way the bot is actually used, in addition to how it is structured.

In this paper, our goal is to consider various bot embeddings and two different but realistic classification tasks, and test whether some bot representations are more appropriate for these tasks. The first task, at the level of entire bots, aims to detect

[*] Work was done while working at IBM.

[1]https://www.techemergence.com/chatbot-comparison-facebook-microsoft-amazon-google/

*Proceedings of the Eighth Joint Conference on Lexical and Computational Semantics (*SEM)*, pages 75–84
Minneapolis, June 6–7, 2019.

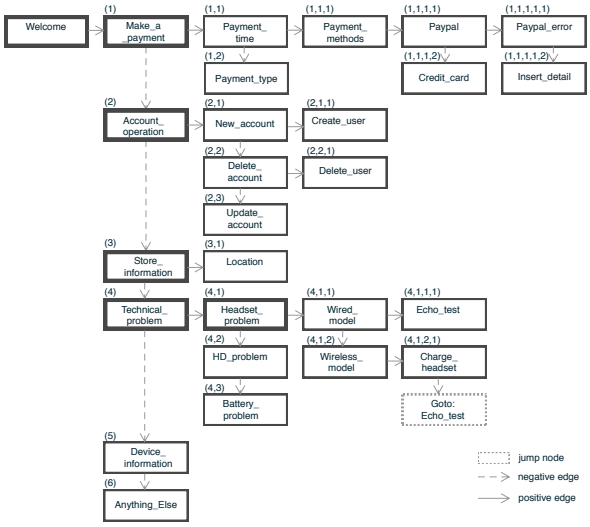

Figure 1: Example of a customer support bot graph.

whether the bot is in production (i.e., in use with real human users) or not. The second task, at the conversation level within each bot, aims to detect problematic conversations with a deployed bot in support of focusing improvement efforts.

The main contributions of this paper are threefold: (1) this is the first research that leverages information from multiple bots in order to improve bot quality, (2) this is the first research to propose an embedding approach for bots based on their structure and how this structure is exploited during conversations, and (3) we empirically evaluate these representations for two classification tasks using data from more than 90 conversational bots.

We find that our proposed representations lead to more than 16% improvement in classification for our two tasks, with our structure-based representation performing better than our content-based representation. This suggests that the representations explored in this paper are valuable across a range of possible tasks.

2 Bot Overview

Although different programming models can be used to create bots, in practice, most commercial conversational system platforms represent the conversation control flow for bots as graphs. In this paper, we use a bot paradigm based on one of the publicly available commercial platforms, but it is quite general and can be adapted to fit to other bot programming platforms. Our example in Figure 1 shows a part of a customer support bot graph, and we use it to explain how such a graph is used in the context of a conversation.

At every step (i.e., every *turn*) of a conversation with a bot, the human user expresses an utterance and the bot analyzes it, determines how to respond and updates its internal state. This determination is executed by traversing the graph, starting from a special node called the *root* node, and moving along the nodes of the graph according to a given set of rules as described below. Note that this description aims to present and explain key abstractions rather than the implementation details of an actual bot platform.

2.1 Graph Components

Every node in the graph has two internal parts: a user intent, and an optional reply of the bot. Given a user utterance, an intent classifier is used to determine whether the user utterance matches the intent associated with the node. For example, the *Technical_problem* node has been defined to capture cases where users encounter a technical problem with a product, and this is what is being expressed in the utterance at hand (e.g., *"I'm having some issues with my headset"*). In this case, the classifier should be able to classify this utterance as relevant to this intent. In practice, the intent classifier is trained from examples of utterances and their corresponding intents, written by bot programmers.

Every node has two optional outgoing edges: a *positive* edge and a *negative* edge. If a user utterance has been classified positively, the optional node answer is presented to the user, and the execution moves to the node which is the target of the positive edge. When the execution moves along a positive edge, it is possible to obtain additional input from the user, and continue the evaluation using this input. If a user utterance has been classified negatively, the execution moves to the node which is the target of the negative edge. As shown in Figure 1, we represent nodes connected by negative edges along the vertical axis, and nodes connected by positive edges along the horizontal axis.

When there is no positive edge and the classification is positive, the execution stops, and the system waits for the next user utterance. When there is no negative edge and the classification is negative, execution jumps back to the root node to resume evaluation from the start.

In addition to the nodes that we just described, there are special *sink* nodes (in our example, the

Agent: *Hi, I'm Eva a virtual assistant, how may I help you?*
Customer: *I'm having some issues with my headset*
Agent: *Which model are you using?*
Customer: *A wireless one.*
Agent: *Please make sure that they are charged.*
Customer: *Yes they are, and they work well with other devices...*
Agent: *Got you. In this case go and try our echo test.*
Customer: *Well, I'll stop by one of your stores and see a technician.*
Agent: *I'm not trained on that yet, but I'm still learning. You may want to rephrase your question and try again.*
Customer: *What is your closest location?*
Agent: *Check here to see our locations in your city.*

Figure 2: Example of a possible conversation.

Anything_Else node), which are not the source of any edges, and which typically trigger a special default message like *"I'm still in a learning phase, is there anything else I can help you with?"*. As bots support only a limited set of intents, this mechanism is used to let the user know that some intent is beyond the knowledge of the bot, and to initiate a recovery process.

2.2 Graph Execution

A conversation starts by traversing the graph from the root node. The root node is special in that it does not expect a user utterance, and it only has a positive edge. Its optional response, which can be a greeting message for example, is only output once at the beginning of the conversation.

Consequently, a user utterance defines a path in the graph, and each conversation between a human and the bot can be represented as a sequence of paths in the graph. Figure 2 shows an example of such a conversation.

The nodes that are evaluated for the first user utterance in Figure 2 (*"I'm having some issues with my headset"*) are marked in bold in Figure 1. Thus, the path that is created by the analysis of this utterance starts with the root node *Welcome*, then moves to the *Make_a_payment* node, checking whether this utterance expresses the user intention to make a payment. Since it is not, control moves to the *Account_operation* node, and then, in turn, to the *Store_information* node, along the negative edges, until it reaches the *Technical_problem* node. Here, the internal classifier determines that the utterance indeed expresses that the user encountered a technical problem. As a result, the control moves along the positive edge to the *Headset_problem* node. Once the node's reply is presented to the user (*"Which model are you using?"*), the system then waits for the next user utterance. The next user utterance (*"A wireless one."*) leads to the *Wireless_model* node, hence the resulting path for this utterance is a continuation of the previous path.

Note that nodes connected vertically by negative edges represent alternative understandings of an utterance. That is, in our example, an utterance can be identified as *Account_operation*, *Store_information* or *Technical_problem*, etc. Nodes connected by horizontal positive edges represent specializations of the analysis. That is, after the utterance is classified as *Technical_problem*, moving along the positive edge will check whether the utterance expresses a *Headset_problem*, or (moving again vertically along negative edges) a *HD_problem* or, alternatively, a *Battery_problem*.

In addition, special *jump* nodes are nodes that allow the conversation to jump to a designated node. In our example the node below *Charge_headset*, that refers to the *Echo_test* node, is a jump. Such jump nodes are not essential, but simplify the graph by preventing duplication of subgraphs.

2.3 Notations

We define the *depth* of the bot graph as the maximum number of nodes from left to right (ignoring the root node), i.e. nodes connected by positive edges. The depth of a node v is defined as the number of positive edges used to traverse the graph from the root node to v. In our example from Figure 1 the depth of the graph is 5, while the depth of *Headset_problem* is 2. We define the level l as the set of all the nodes whose depth is l. We define the *width* of the graph at level l as the maximum number of nodes connected by negative edges at this level. In our example, the width of level 1 is 6, while the width of level 2 is 3.

To further simplify notations, we consider a grid layout that defines coordinates for the nodes from left to right and from top to bottom. For example, node *Technical_problem* is mapped to (4), which means that it is the 4^{th} node from top to bottom at level 1. The node *Headset_problem* is mapped to (4,1), meaning that it is the 1^{st} node at level 2 of the 4^{th} node at level 1. Similarly, the node *HD_problem* is mapped to coordinate (4,2) and *Wireless_model* is (4,1,2). Note that nodes located deeper in the graph are mapped to a longer list of

coordinates. The maximal possible length of a coordinate for a node is the depth of the graph.

2.4 Bot Behavior

The graph of a bot determines its behavior, and thus, the structure of the graph captures interesting properties of the bot. For example, there are bots designed to handle simple Q&A conversations, as opposed to bots that handle filling in the details of complex transactions. For Q&A bots, the graph is likely to be of depth 1, with many nodes at this level, representing various alternative questions and answers. For bots handling complex conversations, the graphs are likely to be deeper in order to handle more complicated cases. In general, bots handling narrow use-cases and which are very specific in their dialog capabilities, are likely to have fewer nodes and more jumps to sink nodes. Thus, in order to capture the bot behavior we should consider the different characteristics of its graph, and this is what we would like to capture in our representation.

3 Bot2Vec Framework

3.1 Representation Learning

In this work, we employ a neural network model to learn the BOT2VEC representation. The training input to this model is either a *content-based* representation or a *structure-based* representation of conversations between a human and a bot. Both are described in the following sections. The result of the training is a vector representation for each bot in the dataset.

To learn this BOT2VEC representation, a fully connected network with N hidden layers is used. During training, the input to this network is the representation of a conversation (either content-based or structure-based), and the ground truth is a one-hot vector of the bot that handled this conversation. In other words, given a conversation c, the network predicts which bot handled c using softmax, that is a distribution over the bots. Thus, the output layer vector of the model has the size of the number of bots in the dataset. Once the model is trained, the representation of a bot b is the weights vector $V_{b\cdot}$, where V is the output embedding matrix (the weights matrix connecting the last hidden layer to the output layer).

The motivation for choosing this representation is that the training procedure (using cross-entropy loss) should drive similar bots to similar representations, given that they handle similar conversations. In the context of learning word representations using the Word2Vec skip-gram model (Mikolov et al., 2013b), the output embeddings were found to be of good quality (Press and Wolf, 2017). Hereafter, we denote the content-based model as BOT2VEC-C, and the structure-based model as BOT2VEC-S.

We now describe how a conversation is represented using its textual content and using its bot graph structure which is used as input for the model.

3.2 Content-based Representation

Conversations between users and bots occur in natural language, and as shown in Figure 2, are composed of user utterances and bot responses. The first step for creating our textual representation of a single conversation is to build a vocabulary, which is the union of all terms across all conversations of all the bots in the dataset. We first mask tokens that might reveal the bot's identity, such as bot names, URLs, HTML tags etc. To neglect additional bot specific words (which are probably infrequent), we take the k most popular terms to be the vocabulary.

Now, for a given conversation, we create two vectors with term frequency (TF) entries according to the template just defined, one for the user utterances and one for the bot responses. The conversation is then represented as their concatenation.

3.3 Structure-based Representation

Our goal in this representation is to characterize bot behavior by analyzing its conversations with respect to the structure of the bot graph. This learned representation should capture the characteristics of how bots are being utilized. We would like to capture, for example, which nodes are being visited during a conversation with a user, at which nodes the conversation turns end, etc.

Recall that each conversation can be represented as a sequence of paths on the bot graph (a path for each turn). We now describe how to represent a path as a vector (*bin vector* below), and how to aggregate paths of a single conversation.

Bin vector To be able to compare bots with different bot graph structures, we define a common fixed size *bin vector* to represent paths of different bots.

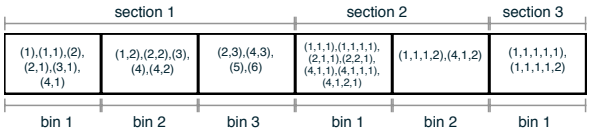

Figure 3: A mapping of a bot graph to the bin vector.

Algorithm 1 Mapping a node to section and bin

input n_l: node depth
n_w: node width
D: graph depth
$w = w_1, w_2, \ldots, w_D$: graph width at each level
S: number of sections
$b = b_1, b_2, \ldots, b_S$: number of bins per section
output n_s: section index
n_b: bin index

1: $n_s = \lfloor \frac{(n_l - 1)}{d} \times S + 1 \rfloor$
2: $n_b = \lfloor \frac{(n_w - 1)}{w_l} \times b_{n_s} + 1 \rfloor$
3: **return** n_s, n_b

We create a bin vector such that each node in the bot graph is mapped to a single bin based on its coordinates in the graph. Each bin vector is divided into S sections, and each section s is divided into b_s bins (Figure 3). Since the idea is to represent a conversation path in a standardized and compact way across different bots, each level in the bot graph is mapped to a section in the bin vector, and each node in the bot graph is mapped to a bin in the appropriate section (see Algorithm 1). Several levels might be mapped to the same section, and several nodes can be mapped to the same bin (sections do not necessarily have the same number of bins). The number of sections and bins in the bin vector are set based on the depths and widths of all the bot graphs (e.g., the average depth of the graphs and the average width of each level in the graphs). For example, the bin vector in Figure 3 represents the mapping of all nodes for the bot graph in Figure 1. This bin vector has 3 sections. There are 3 bins in section 1, 2 bins in section 2, and 1 bin in section 3.

Utterance modeling We now explain how each utterance is represented using the bin vector. As mentioned above, each user utterance in a conversation is represented by a path in the bot graph, whose nodes can be mapped to sections and bins in the bin vector. In order to capture how every utterance is being analyzed by the bot, we distinguish between different types of nodes in the path:

1. A *success (s)* node is the last node of the path, if it is not a sink node.

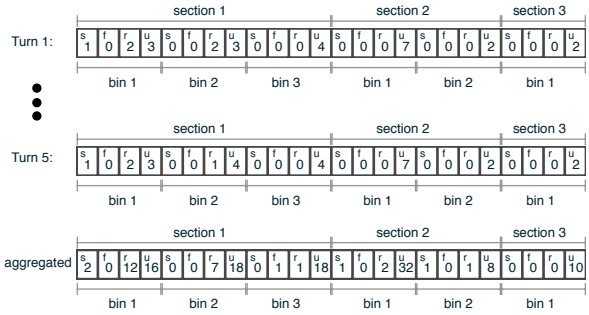

Figure 4: Bin vector representation of turns in a conversation.

2. A *failure (f)* node is the last node of the path, if it is a sink node.

3. All the other nodes that belong to the path are *regular (r)* nodes.

4. Nodes that do not belong to the path are *uninvolved (u)* nodes.

When representing a path, we consider the type of the node it is mapped to, in the corresponding bins in the bin vector: each bin maintains 4 counters, one counter for each type of node mentioned above (*success*, *failure*, *regular* and *uninvolved*). That is, the mapping of the first user utterance *"I'm having some issues with my headset"* to the bin vector is as follows:

- The first node in the bot graph that is visited is *Make_a_payment* (1). This node is mapped to the first bin in section 1 of the bin vector. Thus the *regular* counter is set to 1 for this bin.

- The second node traversed in the bot graph is *Account_operation* (2), which is mapped to the same first bin of section 1 of the bin vector. Hence, the *regular* counter of this bin is set to 2 in the bin vector.

- Similarly, nodes *Store_information* (3) and *Technical_problem* (4) are visited, and that sets the *regular* counter of bin 2 in section 1 to 2.

- Finally, the *Headset_problem* $(4, 1)$ node is visited, and that sets the *success* counter of bin 1 in section 1 to 1, as this is the last node that is being visited for this utterance. Now we can update the *uninvolved* counters of the bins according to the nodes that were not visited during the traversal.

When this path is mapped to the bin vector, we obtain the vector as shown in "Turn 1" of Figure 4.

Conversation modeling As the input to the model is a conversation, we now describe how it is represented. We aggregate the bin vectors of the user utterances paths by summing each counter based on the node types (s, f, r or u) across all the bins in the matching sections. This aggregation captures different patterns of the conversation, such as how many times nodes which are mapped to a bin are visited, how many turns ended successfully in the mapped nodes vs. how many turns failed in these nodes, etc. Figure 4 depicts the detailed vectors for the first and the fifth customer utterance from Figure 2, as well as the aggregated vector obtained for the whole conversation.

4 Classification Tasks

BOT2VEC representations could be used for a variety of bot analytics tasks. In this research, we have examined two such tasks.

Detecting production bots Several companies provide bot development platforms that are used to create and manage conversational bots. Based on analyses of the logs of one commercial platform, we have found that a large percentage of bots are not being used with real users. From the platform provider perspective, understanding bots interaction with actual users could inspire the development of new tools and services that could assist all of bots that use the platform. Thus, it is important to first determine which bots are used in production, rather than in debugging or testing. This is made difficult by the fact that bot testing often involves somewhat realistic simulations of conversations. Thus, in this binary classification task, a bot should be classified as either a production bot or not, given all of its conversations.

Detecting egregious conversations Once in production, bot log analysis forms the basis for continuous improvement. Finding the areas most in need of improvement is complicated by the fact that bots may have thousands of conversations per day, making it hard to find conversations failing from causes such as faulty classification of user intent, bugs in dialog descriptions, and inadequate use of conversational context. Recently, a new analysis (Sandbank et al., 2018), aims at detecting egregious conversations, those in which the bot behaves so badly that a human agent, if available, would be needed to salvage them. Finding these egregious conversations can help identify where improvement efforts should be focused. In this task, a conversation c should be classified as egregious or not, with the BOT2VEC representation potentially improving performance.

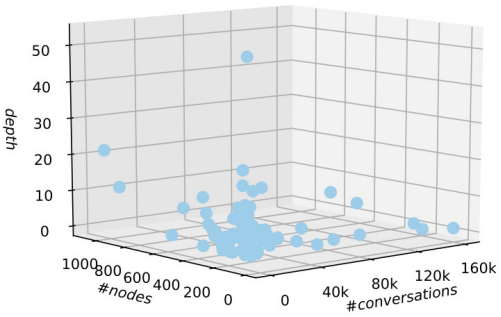

Figure 5: Bots summary: #conversations, #nodes and depth.

5 Experiments

5.1 Data

We collected two months of data from 92 bots, including their graphs and conversations logs. The bot domains included health, finance, banking, travel, HR, IT support, and more. Figure 5 summarizes the information about number of conversations, number of nodes and graph depth for the bots. In total, we collected 1.3 million conversations, with a minimum of 110 conversations and a maximum of $161,000$ conversations per bot. For 62% of the bots, the number of conversations varied between 1000 to $10,000$. Bot graph depth ranged from 2 to 52 levels with an average depth of 7; the total number of nodes ranged from 11 to 1088 with an average of 160 nodes per bot.

5.2 Experimental Setting

Common bin vector As explained, to capture comparable behavior across bots, we create one common bin vector using the average depth and average width for each level of the bots graphs. Specifically, first, based on the average depth, we define the number of sections to be 7. Then we set the number of bins for each section (based on the average width per level per bot) to 108, 10, 6, 17, 8, 4, and 1, respectively.

Bot2Vec implementation details
Content-based: The content-based model input is comprised of two vectors of size $k = 5000$ each ,

Model	F1-score	% improvement
BOT-STAT	0.519	-
BOT2VEC-C	0.545	5.0
BOT2VEC-S	0.616	18.6

Table 1: Production bots detection - classification results on the test set.

Model	F1-score	% improvement
EGR	0.537	-
bot-STAT	0.597	11.0
BOT2VEC-C	0.617	14.8
BOT2VEC-S	0.626	16.4

Table 2: Egregious conversations - classification results on the test set.

one vector representing the user utterances and the second vector representing the bot responses. The two vectors are concatenated and passed through a fully connected layer with 5000 units. We also calculate the squared difference of the two vectors and an element-wise multiplication of the vectors to capture the interaction between the user and the bot. The three vectors are then concatenated and passed through another two fully connected layers with 1000 and 100 units.

Structure-based: The input is a single vector with the size of 616 (the total number of bins (154) times the 4 counters per bin). This input vector is passed through two fully connected layers with 100 and 20 units.

For both models, the last hidden layer is connected to the output layer (with a size equal to the total number of bots). All hidden layers consists of ReLU activation units, and are regularized using dropout rate of 0.5. The models were optimized using an Adam optimizer with a 0.001 learning rate.

5.3 Task 1 - Production Bots Detection

Ground truth To annotate bots, we randomly sampled 100 conversations from our dataset. The sampled conversations were tagged by two different expert judges. Given a full conversation, each judge tagged the conversation as production or test/debugging. If more than 50% of the conversations were tagged as production, then the bot was tagged as production. In addition, if the bot was annotated as not-production, the experts had to provide a list of reasons for their choice (e.g., repeating users ids, repeating bot response, etc.). We generated true binary labels by considering a bot to be a production bot if both judges agreed. Judges had a Cohen's Kappa coefficient of 0.95 which indicates a high level of agreement. This process generated the production bot class size of 40 (44% of the 92 bots).

Baseline model Inspired by (McIntire et al., 2010; Zhang et al., 2018) we implemented a baseline model denoted BOT-STAT as follows: for each bot we calculated features, such as the number of unique customer sentences, number of conversations, number of unique agent responses, and statistical measures (mean, median, percentile) of the following metrics: number of turns of a conversation, number of tokens in each turn in a conversation, and the time of a turn in a conversation. In total we implemented 17 features.

In our implementation we used an SVM classifier (as we only have 92 samples), measured the F1-score of the production bot class, and evaluated the models using 10-fold cross-validation.

Results Table 1 depicts the classification results for the three models we explored. The BOT2VEC-S model outperformed the other models with a relative improvement of 18.6% over the baseline. The performance of the BOT2VEC-C is slightly better than the baseline which indicates that the information that was captured by the content of the conversations was helpful to detect the usage of the bot. The structure-based representation, however, seems to capture bot variability more effectively, i.e. the coverage of visited nodes, different conversations patterns, etc.

5.4 Task 2 - Egregious Conversations Detection

Ground truth To collect ground truth data, we randomly sampled 12 bots (from the production bots), and for each bot 100 conversations were annotated following the methodology in (Sandbank et al., 2018), namely, given the full conversation, each judge tagged whether the conversation was egregious or not. Judges had a Cohen's Kappa coefficient of 0.93 which indicates a high level of agreement. The size of the egregious class varied between the bots, ranging from 8% to 48% of the conversations. All the conversations were aggregated to one dataset.

Baseline model We implemented the state-of-the-art EGR model, presented in (Sandbank et al., 2018). In addition, our models are an extension of the EGR model, such that for each conversation,

its bot representation vector was concatenated to the EGR model's original feature vector.

We measured the F1-score of the egregious class, and evaluated the models using 10-fold cross-validation.

Results Table 2 summarizes the classification results for all models. Specifically, the BOT2VEC-S outperforms all other models with a relative improvement of 16.4%. This suggests that the structure-based representation of the bot encapsulates information which helps the model to distinguish between egregious and non-egregious conversations. Moreover, although the EGR model receives the text of the conversation as input, the content-based representation of the bot also helped to improve the performance of the task.

5.5 Structure-based Analysis

In practice, bots belong to various application domains like *banking*, *IT* and *HR*. Motivated by the semantic similarities between word embeddings (Mikolov et al., 2013b), we further analyzed the structure representation BOT2VEC-S w.r.t bots that belong to the same domain. For the set of production bots, *IT*, *HR*, and the *banking* domains were prominent with 10, 7, and 6 bots respectively, while the other bots belonged to a long tail of domains like *travel*, *medical*, etc. For the prominent domains (that had more than 5 bots), we calculated the average cosine distance between vector representations for pairs of bots that belong to the domain vs. pairs of bots from different domains. We find that the average distance between bots within their domain is 0.614, while the distance between bots from different domains is 0.694. Thus, the representations of bots that belong to the same domain appear to have, as one would expect, a higher level of similarity.

6 Related Work

Despite the popularity of chatbots, research on bot representations and usage analysis is still under explored. Works on chatbot representations are mostly concentrated on neural response generation (Xu et al., 2017; Li et al., 2016; Herzig et al., 2017) and slot filling (Ma and Hovy, 2016; Kurata et al., 2016). In these works, conversation history is used to generate the next bot response. Other works use conversation representations for improving specific tasks useful in dialog, like intent detection (Kato et al., 2017), dialog act detection (Kumar et al., 2018), and improving fluency and coherency (Gangadharaiah et al., 2018). Yuwono et al. (2018) learn the quality of chatbot responses by combining word representations of human and chatbot responses using neural approaches. The main difference between these works and ours is that we analyze multiple bots within a service to generate representations useful to each whereas others analyze a single bot at a time. In addition, we show that our representations are beneficial across different tasks. Finally, none of these works consider the structure of the bot as part of the representation.

Learning embeddings for different objects is one of the most explored tasks[2]. As mentioned above, graphs and nodes representations were proposed in (Narayanan et al., 2017; Grover and Leskovec, 2016). Both works considered static local structures in graphs, whereas our graph representation is based on dynamic conversation paths. The work in (Mikolov et al., 2013b) suggested a Word2Vec skip-gram model such that a word is predicted given its context. In our work, we take a similar approach fitted to the more complex structure of bots, and predict a bot id given a representation of a conversation.

Recently, Guo et al. (2018); Pereira and Díaz (2018) presented a chatbot usage analysis over several bots. Guo et al. (2018) compared the performance of various chatbots that participated in the Alexa prize challenge and implemented the same scenario. To do so, the authors used different measures such as conversational depth and breadth, users engagement, coherency, and more. Pereira and Díaz (2018) suggested a list of quality attributes, and analyzed 100 popular chatbots in Facebook Messenger. Our work focuses on learning bot representations targeted towards classification tasks on the level of bots and their conversations.

7 Conclusions

In this paper, we suggest two BOT2VEC models that capture a bot representation based either on the structure of the bot or the content of its conversations. We showed that utilizing these representations improves two platform analysis tasks, both for bot level and conversation level tasks. Future work includes extension of the model to encapsulate both the content and the structure based

[2]https://github.com/MaxwellRebo/awesome-2vec

representations combined together using sequential neural networks (such as RNN).

References

Rashmi Gangadharaiah, Balakrishnan Narayanaswamy, and Charles Elkan. 2018. What we need to learn if we want to do and not just talk. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, pages 25–32. Association for Computational Linguistics.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM.

Fenfei Guo, Angeliki Metallinou, Chandra Khatri, Anirudh Raju, Anu Venkatesh, and Ashwin Ram. 2018. Topic-based evaluation for conversational bots. *CoRR*, abs/1801.03622.

Jonathan Herzig, Michal Shmueli-Scheuer, Tommy Sandbank, and David Konopnicki. 2017. Neural response generation for customer service based on personality traits. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 252–256. Association for Computational Linguistics.

Tsuneo Kato, Atsushi Nagai, Naoki Noda, Ryosuke Sumitomo, Jianming Wu, and Seiichi Yamamoto. 2017. Utterance intent classification of a spoken dialogue system with efficiently untied recursive autoencoders. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 60–64. Association for Computational Linguistics.

Harshit Kumar, Arvind Agarwal, Riddhiman Dasgupta, and Sachindra Joshi. 2018. Dialogue act sequence labeling using hierarchical encoder with CRF. In *AAAI*. AAAI Press.

Gakuto Kurata, Bing Xiang, Bowen Zhou, and Mo Yu. 2016. Leveraging sentence-level information with encoder lstm for semantic slot filling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2077–2083. Association for Computational Linguistics.

Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org.

Jiwei Li, Michel Galley, Chris Brockett, Georgios Spithourakis, Jianfeng Gao, and Bill Dolan. 2016. A persona-based neural conversation model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 994–1003. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074. Association for Computational Linguistics.

J. P. McIntire, L. K. McIntire, and P. R. Havig. 2010. Methods for chatbot detection in distributed text-based communications. In *2010 International Symposium on Collaborative Technologies and Systems*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005.

Juanan Pereira and Oscar Díaz. 2018. A quality analysis of facebook messenger's most popular chatbots. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163. Association for Computational Linguistics.

Tommy Sandbank, Michal Shmueli-Scheuer, Jonathan Herzig, David Konopnicki, John Richards, and David Piorkowski. 2018. Detecting egregious conversations between customers and virtual agents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1802–1811. Association for Computational Linguistics.

Zhen Xu, Bingquan Liu, Baoxun Wang, Chengjie SUN, Xiaolong Wang, Zhuoran Wang, and Chao Qi. 2017. Neural response generation via gan with an approximate embedding layer. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 617–626. Association for Computational Linguistics.

Steven Kester Yuwono, Wu Biao, and Luis Fernando D'Haro. 2018. Automated scoring of chatbot responses in conversational dialogue. In *Proceedings of International Workshop on Spoken Dialog System Technology*.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing dialogue agents: I have a dog, do you have pets too? *CoRR*, abs/1801.07243.