INTERSPEECH 2016
September 8–12, 2016, San Francisco, USA

Detection of User Escalation in Human-Computer Interactions

Ian Beaver[1], *Cynthia Freeman*[1]

[1]NextIT Corporation, Spokane Valley, WA USA
ibeaver@nextit.com, cwu@nextit.com

Abstract

Detection of virtual agent conversations where a user requests an alternative channel for the completion of a task, known as an escalation request, is necessary for the improvement of language models and for a better user experience. Although methods exist for proactive escalation, we instead wish to explicitly detect escalation requests. In addition, these proactive methods depend on features that do not correlate highly with open-ended chats found in many modern virtual agents. We propose a strategy that can apply to both bounded and open-ended systems since our method has no assumptions on the implementation of the underlying language model. By combining classifiers with several conversation features, we successfully detect escalation requests in real world data.

Index Terms: dialog acts, human-computer interaction, escalation

1. Introduction

We are interested in the automatic discovery of human-computer conversations where users are unable to be assisted by virtual agents. As a company that deploys chatter bots in customer service domains, such conversations provide valuable insight as to where the agent fails so that language model (LM) improvements can be made. One method to label such conversations is to search for turns containing an *escalation request* and determine if the request was made due to dissatisfaction with the agent. We define an escalation request as a dialog act where the user explicitly asks to resolve their issue with a different party than the virtual agent. These conversations are also crucial for the discovery of effective methods to determine when escalation is necessary even when not specifically requested, or *proactive escalation*.

The virtual agents are implemented as intent classifiers and/or task-based dialog managers that typically contain thousands of intents and hundreds of tasks. They are multi-channel in nature and may interact with human users through any combination of websites, mobile apps, social media, or Interactive Voice Response systems. As such, we require a broad method of detection that is not dependant on features from any specific domain or channel.

Currently, a simplistic impasse detection algorithm that looks for the same intent responding repeatedly or recurrent alternating intents is used for detecting agent confusion. While this procedure adequately detects when users reword their questions due to unsatisfactory responses, it does not detect explicit user escalation requests. Our task is to build a system to detect failure-driven escalations that does not depend on any implementation details of the language model or environment. Many approaches to intent classification exist [1, 2, 3, 4], and we assume different implementations may be created based on differing language domain and task.

In this paper, we develop an algorithm that is effective in detecting user escalation language and context. Furthermore, we demonstrate the algorithm's effectiveness using real world chat data.

2. Related Work

Strategies for deciding if or when to escalate exist but typically involve telephone conversations and depend on either the duration of the call or a tree-based dialog structure. Lemon and Pietquin use a classifier to determine if a customer's issue is resolved [5], and a fixed point is chosen at some time in the call for this determination. The duration of the interaction after this point as well as the cost of call parameters are vital in arbitrating whether a human agent is needed. Similar techniques can be found in [6, 7, 8] where classifiers are used to determine if a call is problematic and when an escalation is necessary. Another technique, [5, 9], considers conversations in a tree-based dialog structure. Nodes that have low automation potential have a higher chance of escalation. The call flow is pruned by identifying nodes with low automation potential and eliminating any further flow path following such nodes.

Since our system does not use a dialog tree structure, the latter approaches are not directly applicable. However, a similar feature such as *conversation paths*, which is the sequence of responding intents in a conversation, was examined for its possible relationship with escalation frequency. Conversation paths were extracted from the evaluation dataset described in the following section, and an 80-20 split was done to create training and testing sets. A Naive Bayes classifier from TextBlob, a Python library for processing textual data [10], was trained on the conversation paths. Accuracy achieved on positive escalation turns was extremely low (29%) on test data.

Former methods use conversation duration as a feature; so, we consider duration's relationship to observed escalation frequency in live chat data. However, with a Pearson correlation coefficient of .269 (calculated using the Scipy stats module also on the evaluation dataset described in the following section), duration alone is a poor predictor of escalation. In addition, duration versus escalation frequency does not take into account confounding factors such as user typing speed or multi-tasking. Alternative to duration, one might consider the number of turns in the conversation before escalation. For this, we obtained a Pearson correlation coefficient of .26. Such turn based strategies also do not appear fruitful in detecting escalations.

Although related, these existing methods aim to detect if or when to escalate which is *different* from our task. Our task is not to predict escalation but to detect explicit escalation requests made by the user. More specifically, we wish to detect when the user has *explicitly* requested an escalation *after* first attempting to use the agent. No existing literature could be found on this specific topic. It is clear that in the context of our live

http://dx.doi.org/10.21437/Interspeech.2016-535

chat system, which is turn and not time dependent and also has no tree structure, such traditional escalation strategies are not applicable. Nevertheless, a novel method for detecting a user's request for escalation is necessary since the discovery of conversations where the agent has failed will lead to improvements in the language model. Automatic collection of this data will also generate corpora for investigating more generic methods for proactive escalation than those reviewed.

3. Experiment Setup

For the purposes of our live chat system, we consider escalation requests to be in one of three classes. The first class (I) consists of those users who immediately request to speak to a different party; they do not want to use the automated system. For example:

```
Agent: Hello, how can I help you today?
User: can you transfer me to reservations?
```

The second class (II) of escalations occur after the virtual agent directs the user to contact a different department or service, and, in response, either the user asks to be transferred there or requests contact information. An example of this would be:

```
Agent: To change the name on your ticket, please
       call reservations.
User: can you transfer me to reservations?
```

The final class (III) are those conversations where a virtual agent attempted to resolve an issue for the user, failed to do so, and the user requested an alternative party. For LM improvement, we are not interested in classes I or II since they do not reflect a failure in the conversational ability of the virtual agent. Within our virtual agents, there exists intents to capture various escalation requests and transfer users. However, there are shortcomings in relying only on these intents to surface conversations for review. In our analysis, we observed that escalation requests occur in as many as 16% of conversations. However, the vast majority of them fall into the first two classes, meaning there is no error in the LM to be investigated. Also, we are trying to discover error in the LM itself, so we do not want to rely on it to provide us with instances where it failed. Instead, we develop a stand-alone means to detect only Class III requests for human review and data collection of failed conversations.

We have deployed virtual agents in various domains, but a significant number of them reside in travel-related industries. Therefore, we experiment in this domain as we have access to a large amount of similar data across numerous deployments.

For training, we constructed a set of $15,338$ turns containing $1,703$ manually tagged escalation requests. Due to the expense of humans tagging nearly 45K total turns for train, test, and evaluation, the data was divided equally among the reviewers and combined instead of a more robust multiple review and kappa or majority calculated. The turns were selected from chat histories of US and non-US transportation customer service agents to form a general travel domain corpus. To train a robust binary classifier, we chose a large number of positive examples as escalation language is very broad. The training data contains only the user input and its binary escalation value. We make no distinction in the escalation class for training as the user language is generally the same; it is the context they occur in that differentiates the classes. We develop an effective algorithm to determine the context given a binary classifier that can detect the language.

For development and evaluation, we use raw live chat data collected over a 24-hour period from an airline virtual agent that retains conversation features such as time stamps, conversation ID, responding intent, and response text. These were tagged by the same process, but only Class III escalations were labeled since the ability to discover this type of escalation is what we measure. A small dev-test set was reserved for sanity testing and optimization of vectorization and classifier parameters. It consists of 226 conversations containing 920 turns, of which 29 were Class III escalation requests. The evaluation set is $7,967$ conversations containing $28,336$ turns, of which 333 are Class III escalations. Although only 1-3% of turns in live chats contain a Class III escalation, our virtual agents currently see around 500K inputs daily, and this number is increasing steadily. Thus, we estimate $5,000$ to $15,000$ conversations contain Class III escalations, not an insignificant amount and worthy of investigation.

4. Training an Escalation Classifier

We originally theorized that there were enough differences in the language used for each of the three escalation classes that a multi-class classification approach would be effective. Experimentation revealed that the confusion between the classes was so high that a multi-class approach would not work. Many cases exist where the user inputs are identical but belong in different classes making it impossible for a turn based classifier alone to succeed. An example of this was seen in the previous section, where both user inputs were *"can you transfer me to reservations?"* but did not belong to the same class. Thus, we turn our efforts towards distinguishing the class by conversational context given a binary classifier that can determine the presence of an escalation.

Our first objective is to create a binary classifier that can accurately detect escalation language. As our problem involves text classification, we use a Support Vector Classifier (SVC) with a linear kernel which has been shown to perform well at this task [11, 12, 13]. After every change to the vectorizer and classifier pipeline, we optimize the parameters against the dev-test set and measure performance against the evaluation set. The results of these iterations are shown in Table 1, Methods 1 through 4. Our utility function for optimization maximizes recall on escalation requests. To collect conversations for LM improvement, we err on the side of false positives rather than false negatives. As the evaluation set only tags Class III escalations, we expect the precision to be very low initially as the classifier will consider all three classes of escalations positive. In the following sections, we incorporate this classifier into an algorithm to separate the positive matches into the three classes.

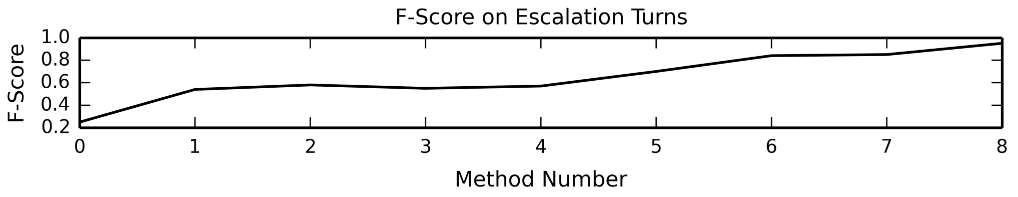
4.1. Baseline for Experiments

As there exists no other escalation detection systems to compare our results against, we create a reasonable first order system to compare to. We take the top 25 most frequent words in the positive training turns and create a Regular Expression (RE) to capture common phrases they appear in. We then run this RE against the evaluation set (Table 1, Method 0) as the baseline to compare our system against. Increased accuracy is possible by combining multiple REs from in-depth analysis, but due to the time consuming nature of constructing manual models, we present the following RE as a reasonable baseline.

```
^.+(t(alk|ransfer)|phone|c(all|ontact)|speak|
need).+(customer(\s+)?(s(ervice|upport))|(a\s+)?
rep(resentative)?|someone\s?else|(a|the|your)\s+
manager|a\s+((live|real)\s+)?(human(\s?being)?|
person)|reservations|number).+
```

Table 1: Algorithm performance in discovery of Class III escalations in live chats

		Class III Escalation Turns		Non-Escalation Turns	
	Method	**Precision**	**Recall**	**Precision**	**Recall**
0	Baseline Regular Expression	0.29	0.23	0.99	0.99
1	Hashing Vectorizer	0.4	0.83	1.0	0.99
2	TF Vectorizer	0.42	0.96	1.0	0.98
3	TF-IDF Vectorizer	0.38	0.98	1.0	0.98
4	POS Tags + TF-IDF Vectorizer	0.4	0.99	1.0	0.98
5	Filter Class I: single turns	0.54	0.99	1.0	0.98
6	Filter Class I: 1..n positives	0.73	0.99	1.0	0.99
7	Filter Class I: initial greetings	0.74	0.99	1.0	0.99
8	Filter Class II	0.9	0.99	1.0	1.0



For our classifier, we initially use SciKit-Learn's HashingVectorizer and LinearSVC to construct a pipeline [14]. The SVC is trained with the default parameters, and we use a grid search against the dev-test set to find optimal parameter values. These parameters and their function are described in [15] and disscussed at length in [16]. The optimal parameters found were *C=2.1, penalty=l2, loss=squared_hinge* (Table 1, Method 1).

4.2. Methods for Classifier Improvement

This classifier only produces 83% recall, so we focus on feature improvement (Table 1, Methods 2-4). A basic first step is to incorporate raw term frequency for term weighting and N-grams to include context. Changing to SciKit-Lean's TfidfVectorizer, we optimize both the SVC parameters and the lower and upper range boundary of n-words. The additional optimal parameters were *C=1.7, ngram_range=(1,3)* (Table 1, Method 2).

To further isolate language specific to escalations, we turn on Inverse Document Frequency (IDF) to scale down the impact of words that are less informative [17, 18]. The additional optimal parameters were $C=2.5$, *ngram_range*$=(1,4)$ (Table 1, Method 3).

Including Part Of Speech (POS) tags can benefit SVC for text categorization [19]. So, we add POS tagging as a preprocessing step. Using the Natural Language ToolKit (NLTK) PerceptronTagger [20], we add the POS tag to each word in the input text and discard all punctuation before feeding the text into the pipeline (Table 1, Method 4).

4.3. Rejected Methods

We also considered the following methods and determined that they did not significantly increase performance over Method 4.

4.3.1. Automatic Spelling Correction

Using the TextBlob *correct* method [10], which replaces words with the highest probability of known words with edit distance 1-2 from the given word, we observed a minor increase in false negatives and a 0.5% decrease in false positives. The gains were not high enough to justify the additional complexity, but it remains an interesting possibility. There are much better means of automatic spelling correction that take into account context [21, 22] and applying them is subject to future research.

4.3.2. Stemming

Using the NLTK English SnowballStemmer [23], we stem the words after generating the POS tag and replace the original word with its stem. In agreement with the conclusions of [24], stemming introduced ambiguity in the SVC that led to a loss of performance. We observed a negligible increase in false negatives and a 10% increase in false positives.

4.3.3. Stopword Filtering

Using the NLTK English stopword list, we filter all stopwords before the POS tagging. Also in agreement with [24], we found that the removal of stopwords was unnecessary for SVC accuracy. Stopword removal resulted in a minor increase in false negatives and a 12% increase in false positives.

4.3.4. Sentiment Analysis

Using the NLTK NaiveBayesAnalyzer [25], which is trained on the Movie Reviews corpora, we add turn sentiment polarity and subjectivity as two additional feature columns to the feature vector produced by TfidfVectorizer. By adding them to the feature vector, we allow the SVC to discover if any correlation is present. This resulted in no change to false negatives and a negligible increase in false positives. As there exists better methods for sentiment classification ([26], for example) and these scores can be integrated in other ways, we will revisit sentiment analysis in future work. We did observe, however, that despite users feeling frustrated while the agent is misunderstanding them, many of them still politely request an escalation. This makes it difficult to rely on sentiment polarity as a feature of an escalation request.

5. Algorithm for Escalation Categorization

Using the escalation classifier in Table 1, Method 4, we now focus on precision by designing an algorithm that can separate the positive matches into the three classes (Methods 5-8). We begin with filtering Class I escalations, where the user has no intention of using the agent; they simply request contact information or immediate transfer.

If the initial turn is an escalation, three directions can be taken on the next turn: the conversation ends, the user is not satisfied with the response and rewords his or her question or asks for additional clarification, or the user decides to attempt to resolve their issue with the agent and states their concern. To eliminate the first case, we ignore all conversations with only a single turn. This results in a 43.2% drop in false positives as seen in (Table 1, Method 5).

We handle the second case by feeding the remaining turns through the SVC. We ignore every sequential positive match until either a negative match occurs or the conversation ends. We can then remove all conversations containing only escalation language, and if the user decides to use the agent after all, we consider any escalation occurring after the user identifies their issue as a Class III. This results in a further 32.4% drop in false positives (Table 1, Method 6).

In reviewing the evaluation data, we noticed conversations where the first turn consisted solely of a greeting (*"hows it going?", "Good morning."*). These add noise to the above mentioned Class I detection and should be ignored. To remove them, a separate binary SVC is trained on a dataset created for this purpose. As all of our virtual agents have a *Hello* intent for responding to greeting language, we gathered 226 unique user inputs from across several US domestic and international deployments that were assigned to this intent. International and US domestic greetings are mixed to make the classifier more robust. We then added 59,960 turns assigned to other intents from the various deployments as negative examples. This classifier achieved 0.986 precision and 0.89 recall on the first turn of the 7,967 conversations in the evaluation set. Using this classifier to ignore greetings in the fist turn further reduced the false positives by 2% (Table 1, Method 7). This proved to be a minor improvement, however, and can be considered an edge case.

Next, we focus on filtering Class II escalations. With each turn, we have the responding intent and the response text. Therefore, if a turn is positive for escalation, we ignore it if the previous intent was in a set of escalation intents. This ignore set was created by reviewing all agent responses and collecting intents responding with instructions to escalate the issue. Leveraging this conversational context removes 73% of the remaining false positives (Table 1, Method 8).

Any remaining escalations are considered members of Class III. The final algorithm using both the classifier and set of intents to ignore is given in Algorithm 1.

Algorithm 1: Detect escalations and categorize

```
def LabelConv(conversation):
    Data: A conversation as an ordered list of turns
          containing (input text, responding intent)
    Result: The conversation with labeled turns
    if length(conversation) $< 2$:
        /* Filter single turns */
        label turn as false;
        return conversation;
    skip = true;
    for turn in conversation:
        tagged_text = POS_tag(turn.user_input);
        feature_vect = vectorizer(tagged_text);
        if hello_clf.predict(feature_vect) $== 1$:
            /* ignore greetings */
            label turn as false;
            continue;
        if esc_clf.predict(feature_vect) $== 1$:
            if skip:
                /* ignore 1..n Class I */
                label turn as false;
                continue;
            if prev_turn.intent in ignore_intents:
                /* ignore Class II */
                label turn as false;
            else:
                /* Class III */
                label turn as true;
        else:
            label turn as false;
        skip = false;
    return conversation;
```

6. Conclusions

The use of virtual agents in managing our daily tasks will only increase [27]; therefore, automatic failure detection is essential. While methods for proactive escalation detection exist, they do not detect explicit requests for escalation. Although the analysis of this former type of escalation can improve user experience, there exists a real need to detect explicit escalation requests due to agent failure. In addition, existing methods depend on features such as conversation duration which do not correlate well with open-ended chats or multi-channel environments. To research more effective methods of detecting proactive escalation for these environments, the automatic collection of training data is of utmost necessity. Of the 500K inputs our agents currently see on a daily basis, as many as 15K conversations contain Class III escalations. Our algorithm provides an efficient means for collecting this data.

When applied to a live system, our method can be used to flag conversations in real time for review, regardless of the channel the agent is deployed on. It can also be used for a more intelligent transfer to a live agent. For example, the agent can use an apologetic tone to reduce user frustration or warn the receiving live agent that the user has had a bad experience. Detecting these conversations is essential for LM improvements by collecting failure scenarios independent of subject.

It is clear that a strategy for detection that can apply broadly to a wide variety of systems is of necessity. We make no assumptions on the underlying implementation; therefore, our algorithm can be applied to conversational data from both bounded and open-ended systems regardless of channel. Our results demonstrate that by combining classifiers with language features and narrowing our search to a specific class of escalations, we can successfully detect failure-driven escalation with high accuracy on real world data.

Open problems remain such as additional analysis on feature selection to prevent false positive escalations. Although word-based automatic spelling correction did not improve performance, experimenting with alternative context-based spelling correctors, adding speech frames [28, 29], and running the algorithm on a larger evaluation set would be informative. This paper has, however, presented an effective algorithm for detection of failure-driven escalation requests that can be broadly applied.

7. References

[1] P. R. Cohen, J. L. Morgan, and M. E. Pollack, *Intentions in communication*. MIT press, 1990.

[2] K. A. Tahboub, "Intelligent human-machine interaction based on dynamic bayesian networks probabilistic intention recognition," *Journal of Intelligent and Robotic Systems*, vol. 45, no. 1, pp. 31–52, 2006.

[3] T. Holtgraves, "Automatic intention recognition in conversation processing," *Journal of Memory and Language*, vol. 58, no. 3, pp. 627–645, 2008.

[4] C. A. Montero and K. Araki, "Enhancing computer chat: Toward a smooth user-computer interaction," in *Knowledge-Based Intelligent Information and Engineering Systems*. Springer, 2005, pp. 918–924.

[5] O. Lemon and O. Pietquin, *Data-Driven Methods for Adaptive Spoken Dialogue Systems: Computational Learning for Conversational Interfaces*. Springer New York, 2012. [Online]. Available: https://books.google.de/books?id=d9VmX_zZuSAC

[6] M. Walker, I. Langkilde, J. Wright, A. Gorin, and D. Litman, "Learning to predict problematic situations in a spoken dialogue system: experiments with how may i help you?" in *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics, 2000, pp. 210–217.

[7] E. Levin and R. Pieraccini, "Value-based optimal decision for dialog systems," in *Spoken Language Technology Workshop, 2006. IEEE*. IEEE, 2006, pp. 198–201.

[8] W. Kim, "Online call quality monitoring for automating agent-based call centers," in *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[9] T. Bosse and S. Provoost, "On conversational agents with mental states," in *Intelligent Virtual Agents*. Springer, 2015, pp. 60–64.

[10] S. Loria, "Textblob: simplified text processing," *Secondary TextBlob: Simplified Text Processing*, 2014.

[11] T. Joachims, *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[12] M. Ikonomakis, S. Kotsiantis, and V. Tampakas, "Text classification using machine learning techniques." *WSEAS Transactions on Computers*, vol. 4, no. 8, pp. 966–974, 2005.

[13] T. Joachims, *Text categorization with support vector machines: Learning with many relevant features*. Springer, 1998.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] Various, "sklearn linearsvc documentation," 2016, available online at http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html.

[16] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[17] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, 2003.

[18] D. Hiemstra, "A probabilistic justification for using tf× idf term weighting in information retrieval," *International Journal on Digital Libraries*, vol. 3, no. 2, pp. 131–139, 2000.

[19] H. Taira and M. Haruno, "Feature selection in svm text categorization," in *AAAI/IAAI*, 1999, pp. 480–486.

[20] M. Honnibal, "A good part-of-speech tagger in about 200 lines of python," spaCy Blog, 2013, available online at https://spacy.io/blog/part-of-speech-POS-tagger-in-python.

[21] M. A. Elmi and M. Evens, "Spelling correction using context," in *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 360–364.

[22] M. Schierle, S. Schulz, and M. Ackermann, "From spelling correction to text cleaning–using context information," in *Data Analysis, Machine Learning and Applications*. Springer, 2008, pp. 397–404.

[23] M. F. Porter, "Snowball: A language for stemming algorithms," 2001.

[24] E. Leopold and J. Kindermann, "Text categorization with support vector machines. how to represent texts in input space?" *Machine Learning*, vol. 46, no. 1-3, pp. 423–444, 2002.

[25] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[26] X. Ding, B. Liu, and P. S. Yu, "A holistic lexicon-based approach to opinion mining," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 2008, pp. 231–240.

[27] H. P. Levy, "Gartner predicts our digital future," 2015, available online at www.gartner.com/smarterwithgartner/gartner-predicts-our-digital-future/.

[28] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 86–90.

[29] D. Das, D. Chen, A. F. Martins, N. Schneider, and N. A. Smith, "Frame-semantic parsing," *Computational Linguistics*, vol. 40, no. 1, pp. 9–56, 2014.