Intent-driven Command Recommendations for Analysts

Rohin Garg[*]
IIT Kanpur
Kanpur, India
sronin@iitk.ac.in

Samarth Aggarwal[*]
IIT Delhi
Delhi, India
cs1160395@cse.iitd.ac.in

Abhilasha Sancheti
Adobe Research
Bangalore, India
sancheti@adobe.com

Bhanu Prakash Reddy Guda
Adobe Research
Bangalore, India
guda@adobe.com

Iftikhar Ahamath Burhanuddin
Adobe Research
Bangalore, India
burhanud@adobe.com

ABSTRACT

Recent times have seen data analytics software applications become an integral part of the decision-making process of analysts. The users of these software applications generate a vast amount of unstructured log data. These logs contain clues to the user intentions, which traditional recommender systems may find difficult to model implicitly from the log data. With this assumption, we would like to assist the analytics process of a user through command recommendations. We categorize the commands into software and data categories based on their purpose to fulfill the task at hand. On the premise that the sequence of commands leading up to a data command is a good predictor of the latter, we design, develop, and validate various sequence modeling techniques. In this paper, we propose a framework to provide intent-driven *data command* recommendations to the user by leveraging unstructured logs. We use the log data of a web-based analytics software to train our neural network models and quantify their performance, in comparison to relevant and competitive baselines. We propose a custom loss function to tailor the recommended data commands according to the intent provided exogenously. We also propose an evaluation metric that captures the degree of intent orientation of the recommendations. We demonstrate the promise of our approach by evaluating the models with the proposed metric and showcasing the robustness of our models in the case of adversarial examples, where the intent input is misaligned with the user activity.

KEYWORDS

Command recommendation; intent modeling; user behavior; application logs; topic modeling

ACM Reference Format:
Rohin Garg, Samarth Aggarwal, Abhilasha Sancheti, Bhanu Prakash Reddy Guda, and Iftikhar Ahamath Burhanuddin. 2020. Intent-driven Command Recommendations for Analysts. In *Proceedings of the 2020 World Wide Web Conference (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

[*]Both authors contributed equally to this research.

1 INTRODUCTION

There has been tremendous growth in the domain of data analysis as the volume of data has increased, and as the capabilities to support processing of this data have advanced. Analysts, and more generally users of data-centric software, need to make several selections within the software application to achieve certain objectives, gather insights from the data and take downstream decisions [39]. Considering the sheer volume of data to be analysed, there is now a demand on systems to query, analyze and draw inferences with a low latency. This demand also carries over to users assigned with the task of analysing the data. Recommender systems have been used time and again in a variety of different applications to guide the users. They solve two major concerns:

(1) When a user is faced with a raft of different options to choose from, recommender systems act as a primary filter, weeding out options that are completely irrelevant. This leaves the user with a choice among a relatively smaller number of options [36].
(2) When a novice user lacks the skills and knowledge to choose from the different options provided, recommender systems act as a guide for the user in making a selection [7, 15]. We refer to this click as a *command* or *action* that gets registered in the log data when a user interacts with the interface of the system.

The domain of analytics offers both these problems simultaneously, and our solution approach proposes to build a system that caters to both of them. Introducing recommendations to help the analysts decide what activity to perform and where to look in the data to discover insights can boost productivity. However, recommendations can be distracting if they are irrelevant to the analyst's intent and do not provide guidance towards the insights that the analyst needs.

Human activity is often driven by intent [44]. Natural language phrases indicate the intent of the user when interacting within a help section of a software application, with search engines, or with chatbots [4]. On the other hand, recommendation applications work in the absence of such explicit cues from the user. The interaction mechanism illustrated in this paper borrows the notion of intent and applies it to software assistance that is provided by a recommender system. Specifically, the intent provided as input by the user during a session of interaction with a software application guides the recommendations the user receives. Our investigation is under the purview of an interactive data analysis and visualization software for web analytics. We believe that claims and results presented here generalize to assistance in software applications in similar domains. Our contributions in this work are summarized below.

(1) We conceptualize the notion of intent and propose models where intent is provided exogenously. We design, develop and evaluate sequence-based intent-driven models, which are better predictors of *data commands* compared to traditional recommender systems that were unaware of the user's intent. We demonstrate the performance of our approach through offline evaluations.
(2) We design a custom loss function for models which steer the recommendations towards the user's intent. The loss function involves a probability distribution component, which is different for each of the identified intents, thereby generating an ensemble of fine-tuned models.
(3) We also introduce a measure to evaluate the degree of intent orientation in recommendations provided by the proposed models.

To demonstrate the robustness of our model, we present the performance of our models under adversarial examples. The results showcase how our models would operate under misspecified information when intent input is misaligned with user's activity.

The paper is organized as follows. We present related work in Section 2. In Section 3, we describe the dataset used in our experiments. Sections 4 and 5 describe our intent-driven models and baselines, and experimental results respectively. Next in Section 6, evaluation metrics and results are showcased.

Notation We define $[a, b]$ to denote the sequence of integers a through b. The set of real-valued vectors of dimension d is denoted by $\mathbb{R}^d$. Matrices are denoted by upper case letters in a bold font.

2 RELATED WORK

User intent in the domain of search systems is well-studied with respect to search queries [18] and click behaviours [48] are analyzed. There is limited work on intent in the context of recommender systems. In [5], a system which provides recommendations based on inferred intents is presented. Recently, the role of user intent in a music streaming app was studied in [26]. Intents were identified through face-to-face interviews and via a large scale in-app survey. They proceed to investigate the interplay between interaction and intent in predicting user satisfaction. Tao et al. describe a neural network model termed *Log2Intent*, which infers user interest tags, which are referred to as intents. These tags serve as labels for the multi-label classification using data from user logs and a tutorials; the latter includes human annotations for the action sequence in the tutorials [41]. In contrast, our approach of inferring intents is unsupervised and the intents are exogenously provided as input to the command recommendation model.

Intent-driven recommendations: While a significant amount of research has been carried out to understand and analyze patterns in user behavior from application log data [1, 10, 11, 23], limited research has been published to explicitly incorporate intent information to recommend future commands [5, 27, 29, 45]. The paper [22] models the workflows in event log data based on probabilistic suffix trees (PST) and [23, 45] deploy the techniques of topic modeling to determine the workflows or tasks. In [45], a system to recommend video tutorials based on the command logs from the user activity is discussed. Here a hierarchical approach that operates both at the intent and command level is proposed. The authors consider topics determined through a topic model as intents and at the command level they utilize frequent pattern mining and itemset mining techniques to extract frequent command patterns for videos. [45] build their approach on top of a study of frequent user tasks from product log data [11]. Additionally, they leverage topic modeling as an antecedent layer to capture diverse behavior patterns. This modification has allowed them to represent command patterns from less frequent tasks more faithfully. Elsewhere the concept of context is captured by representing the sessions through trees [27].

Liu [23] explored the problem in the same direction as [45]. Unlike the latter approach, they intend to model the workflows using probabilistic suffix trees. In a similar vein, [29] have proposed neural network architectures *TaskRNN* and *JTC-RNN* to predict the next command in the sequence. Their architecture is similar to that of *TopicRNN* [12] from the domain of text analysis. In line with prior work [23, 29, 45], we use topic modeling techniques to model intents that users carry out by executing a sequence of commands. As topic modeling techniques such as Latent Dirichlet Allocation (LDA) [8] suffer from data sparsity issues, we have used Biterm Topic Modeling (BTM) [46] to identify intents from the log data as typical user sessions are small in size. We define the intents mathematically based on the outputs of the BTM model. Researchers have proposed architectures on incorporating the document context in the form of topic-model like architecture, thereby granting a broader document context outside of the current sentence [21]. Inspired by this language modeling architecture, we propose methods to incorporate current intent information for predicting the next *data command* in the sequence. One of the major drawback of these approaches is that they utilize standard cross entropy loss function which does not consider any intent information while penalizing their models. To the best of our knowledge, there has also been no work which introduces a loss function to steer the recommendations based on the intent under consideration.

The Fine-tuning paradigm: Our proposed models employ the technique of fine-tuning inspired by the Generative Pre-Training (GPT) [34], Generative Pre-Training 2 (GPT-2) [35] modes of training. Both GPT and GTP-2 models demonstrated large gains on natural language understanding tasks by pre-training of language models on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task. The GPT and GPT-2 models utilizes task-aware inputs during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. In this paper we apply a similar framework for the first time to provide intent-driven recommendations by initializing parameters from a trained model for generalized recommendations and fine-tuning with custom loss function to achieve intent specific models. This is very effective in providing recommendations where the user activity is misaligned with the intent given as input.

3 DATASET

The application under consideration is a web-analytics system used to track, report, analyze, and visualize web traffic. Traces of user activity within this software is available as log data. This log data contains the user interface clicks that are captured while the user interacts with the system. We term these clicks as commands, and they fall into the following two categories (1) Software Commands; and (2) Data Commands. The first category of commands is related to analyzing data by the user through mental effort via tables and visualizations, or by applying software tools with underlying statistical or machine learning models. This set of commands is denoted by SC for *Software Commands*. The commands which specify the data to be analyzed fall into the second category. This second set of commands is denoted by DC for *Data Commands*. We observe that selection of these categories of commands is iterative in nature. We would like to provide examples from a familiar framework — spreadsheet applications. In spreadsheets, the software commands refer to opening, loading or saving a spreadsheet, changing the color of a cell, whereas data commands include making column selections, sorting a data column in an increasing or decreasing order, computing aggregation functions such as sum or maximum or minimum of a data column.

Often, an analyst begins analysis with the intent to provide explanations for certain patterns which are captured in the data. The software commands merely facilitate data analysis. On the other hand, the answers sought by the analyst are closely related to the data that is being analyzed. Our objective is to build a system to assist the user by providing guidance on where to look at in the data. Therefore, out of these two sets of commands, we would like to predict only the data commands. After pre-processing three months of usage logs from April 2018 to June 2018, we have extracted data for a few hundred users in the specified duration. The logs with sparse user activity were dropped. The extracted logs were then split into sessions[1] considering the *sign-out* software command as the delimiter. To handle the cases where the user does not end the session explicitly by issuing the *sign-out* software command, we have considered six hours of inactivity as a signal for the end of the session. The duration of inactivity we chose is well beyond the time after which the application terminates the session automatically, if no user response is recorded. The session thus captured were $55K$ in number. There were around 838 unique commands, which were obtained after dropping the commands that were logged to indicate user interace events and not explicitly executed by the users. Out of these 838 commands, 766 commands were the data commands (DC), and the rest of them were the software commands (SC). There is a limited amount of our subjective judgement to arrive at this categorization. Beyond this 838 set of commands, every other recorded click is either a user interface event by the application or by the user which is beyond the scope of the task at hand.

After the pre-processing phase, the average number of commands per session was found to be 30. For uniformity, sessions with length less than 30 were dropped, and sessions with length greater than 30 were handled based on sliding window approach

[1]A session is a sequence of commands executed by a single user from entering the application to exiting the application.

$$\boxed{SC, DC, DC, SC, ..., SC,}\ SC, SC, \mathbf{DC}, DC, SC, ..$$

Figure 1: In this example session of user activity, sequence data is generated for model training by placing a window at each position in the session. The next command that needs to be predicted for the sequence in the window is the immediate data command after the window ends (command in bold).

which is explained in Figure 1. A total of $140K$ sequences, thus generated, were split into training and test sets on the basis of sessions, with roughly 20% in the latter set.

4 INTENT-DRIVEN MODELS

The interaction in our proposed system has the user selecting intents represented by phrases at the start of a session in line with their intended task. We reiterate that the validation of this proposal is through offline evaluation as described in Section 6. Example phrases for intents for the dataset under consideration are also provided in that section.

To recommend data commands based on the intent provided by the user, we propose variants of sequence to sequence (seq2seq) models which are compared against more traditional approaches such as popularity based models, bag-of-commands models, and Markov models. However, before recommending the data commands, the first sub-problem is to identify the possible intents from the user log data.

We handle each of the command sequences, which are obtained from usage log data, as a document and train a BTM [46]. We use BTM, instead of using more popular approaches like LDA, to alleviate the data sparsity problem. This problem arises due to co-occurrence matrix for each and every pair of commands being very sparse. Following this, we use the obtained command distribution $P(data\ command\ (dc)\ |intent\ =\ I)$, for each intent I, to provide human-readable descriptions for the intents obtained through topic modeling. The intent information is delivered to the seq2seq models using two approaches by (1) providing data that corresponds to the intent, and (2) by building intent informed models. In the former approach, we implicitly incorporate the intent information through the data, while in the latter approach, we explicitly input the intent information to the model. We start this section by describing the methodology of identifying the intents, followed by a description of intent specific data models and intent informed models. We then propose a loss function that improves the performance of the intent informed models when evaluated with our proposed metric, which measures the degree of intent orientation.

4.1 Intent Identification

The problem of making recommendations intent aware requires solving two problems, intent identification and adding intent information to the recommender system. The former is a pre-requisite for the latter and was done from user log data, though does not affect the latter.

In this section, we draw an analogy between a collection of natural language documents and a collection of command sequences. A word, which is a fundamental unit of a document corresponds to a command in sequence. Unlike words, in our setting there are two categories of commands, data commands (DC) and software commands (SC) as described in Section 3. The words constitute the documents and similarly the commands ($DC \cup SC$) form the sequences. With this analogy let us describe the process of the BTM in our setting. The generative process of the BTM is described in [46] as follows:

(1) For each intent I
   (a) draw a intent-specific command distribution $\phi_I \sim \text{Dir}(\beta)$
(2) Draw an intent distribution $\theta \sim \text{Dir}(\alpha)$ for the whole log data.
(3) For each biterm b in the biterm set B (set of the biterm commands)
   (a) draw an intent assignment $I \sim \text{Multi}(\theta)$
   (b) draw two commands $c_i, c_j \sim \text{Multi}(\phi_I)$

where α and β are the Dirichlet priors, Dir denotes the Dirichlet distribution, and Multi represents multinomial distribution. The values of the α and β determine the output distributions of the intents. Low values of α and β will give rise to sharper distributions whereas the other extreme outputs flat distributions. The command distribution, ϕ_I, contains the probability values for all the commands, software and data commands. As we aim to predict only the data commands, those probabilities that pertain to the data commands are considered and normalized to generate a distribution for the intent I.

$$P(dc = dc_i|intent = I) = \frac{\phi_I[dc_i]}{\sum_{c \in DC} \phi_I[c]} \tag{1}$$

where $dc_i \in DC$ is the i^{th} data command, $\phi_I[c]$ denotes the probability of the command c in the command distribution for the intent I. The computed distribution, $P(dc|intent = I)$, is considered as the definition for the intent I.

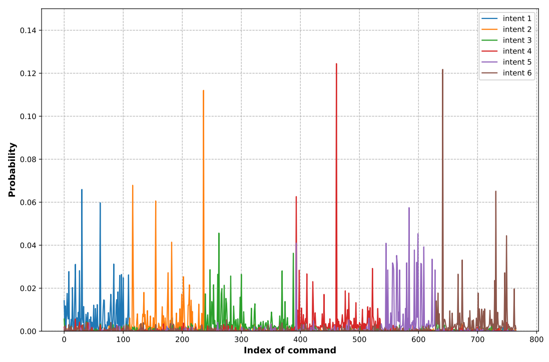


Figure 2: The probability distribution of commands for each of the identified $K = 6$ intents: $P(dc|intent)$. The commands are grouped together intent-wise. This graph illustrates the distinguishable definitions of intents through their probability distributions

The hyperparameters of the BTM model in our setting are the number of intents $K, \alpha,$ and β. To come up with coherent number of intents, one has to go through each of the $140K$ training samples, manually annotate each instance with either one of the previously identified intents or an entirely new intent. The identified intents are said to be coherent if, given any 2 data commands from a cluster (intent), they should be highly similar to each other, and data commands from different clusters should be highly antithetical to each other. Thus the coherence score assesses the quality of the learned intents. Since this process is a very tedious task, we propose an alternative approach that automates this task to an extent. This technique involves experimenting with various number of intents for the log data, and based on some evaluation metrics, the number of intents is finalized. The evaluation metrics used for computing the coherence of identified intents are explained in the following section.

4.2 Intent Coherence

Perplexity is a measurement of how well a probability distribution or probability model predicts a sample. The judgment by a human evaluator not being correlated to perplexity is the motivation for more and more work trying to model the human judgment. This is by itself a hard task as it is difficult to clearly define a human judgment; for example, two experts of the web-analytics software can disagree on the usefulness of a particular intent. As an early evaluation method, [2] define an unsupervised intent ranking measure based on three prototypes of irrelevant and insignificant intents. The three prototypes for irrelevant intents are the uniform command-distribution, $P(dc|intent) \propto 1$, the empirical corpus intent-distribution $p(dc|intent) \propto count(dc \text{ in dataset})$, and the uniform document-distribution $p(session|intent) \propto 1$. Then, an intent significance score is computed by applying various similarity measures such as cosine, correlation and dissimilarity measures such as KL divergence [20] to these three prototypes. It is unclear to what extent their unsupervised approach and objective function agrees with human judgements, however, as they present no user evaluations [30]. Consequently, more and more works have been done to correlate the evaluation measure with human judgement. Two such state-of-the-art intent coherence (topic coherence) evaluation measures are the UCI and UMass. Both the measures compute scores for all pairs of data commands in an intent cluster. They differ in defining the score for a pair of data commands.

4.2.1 ***UCI measure.*** The UCI measure [30] uses pairwise score function, Pointwise Mutual Information (PMI) to compute the score between two data commands.

$$score_{UCI}(dc_i, dc_j) = \log \frac{p(dc_i, dc_j)}{p(dc_i)p(dc_j)} \tag{2}$$

where $p(dc_i)$ represents the probability of seeing data command dc_i in a session, and $p(dc_i, dc_j)$ is the probability of observing both dc_i and dc_j co-occurring in a session. Though [30] proposes to use an external corpus such as Wikipedia to compute the observation probabilities, we use the original dataset of sessions to compute the probabilities making the evaluation measure intrinsic. Therefore, the $p(dc_i)$ and $p(dc_i, dc_j)$ values are computed using

$$p(dc_i) = \frac{M(dc_i)}{M} \text{ and } p(dc_i, dc_j) = \frac{M(dc_i, dc_j)}{M} \tag{3}$$

where $M(dc_i)$ is the count of sessions containing the command dc_i, $M(dc_i, dc_j)$ is the count of sessions containing both commands dc_i and dc_j, and M is the total number or sessions. The UCI score for an intent cluster, I, is computed as mean$\{score_{UCI}(dc_i, dc_j), dc_i, dc_j \in I, i \neq j\}$. The overall UCI score equals the mean UCI score of all the intents. The overall UCI score for a model with t intents, $CS_{UCI}(t)$ equals the mean UCI score across those t intents.

4.2.2 *UMass measure.*

The UMass measure [28] uses a pairwise score function similar to UCI.

$$score_{UMass}(dc_i, dc_j) = \log \frac{M(dc_i, dc_j) + 1}{M(dc_i)} \tag{4}$$

The overall UMass score, CS_{UMass}, is computed similar to the UCI measure. This score function is not symmetric due to the nature of the denominator and therefore, the order of the arguments matters. It is an increasing function of the empirical probability $p(dc_j|dc_i)$, where dc_i is more common than dc_j, commands being ordered by decreasing frequency $p(dc|intent)$. Consequently, this score measures how much, within the data commands used to describe an intent, a common data command is in average a good predictor for a less common data command. The intuition behind using the UMass measure in our setting is as follows. The denominator is the number of sessions c_i appears in. Therefore, the score is higher if dc_i and dc_j appear together in sessions frequently relative to how often dc_i alone appears in sessions. This makes sense as a measure of intent coherence, since if two data commands in an intent really belong together we would expect them to show up together very frequent. The same intuition applies to UCI as well. And hence, higher UCI and UMass scores indicate better groupings. The process of choosing optimal number of intents for our data by considering both these measures is discussed in Section 5.

4.3 The Ensemble Approach

The information of intent can be implicitly incorporated into the models through the data that is provided as input to them. Several applications have used this approach earlier to generate models specific to a data distribution observed in a particular class or task in general. Multi-class classification [24, 25, 33, 37], is a classic example of this approach where the data distribution of a class is assimilated through an ensemble of models. Similar to this approach, we propose one model per intent, which is explicitly trained to model the distribution of the data of that particular intent. The technicalities of the model are as follows. The corpus is split into K mutually exclusive and exhaustive sets based on the output of the trained BTM model. The BTM model assigns an intent distribution for each command sequence which is a probability distribution over the identified K intents. Each command sequence is labeled with the intent that has the highest probability in its assigned intent distribution. Our proposed models use multi-layered Long Short-Term Memory (LSTM) [16] to encode the input sequences of commands into vectors of fixed dimensionality. Given a sequence S with $c_i \in$ DC $\cup$ SC, i in $[0, L]$ commands, we first embed the commands through an embedding matrix $\mathbf{W}_e$ such that $x_i = \mathbf{W}_e c_i$. This sequence of embedded commands is then provided as an input to an LSTM encoder which computes representations of the commands by summarizing the information. The representation from the last time step of the LSTM hidden unit, v_L, denotes the semantic representation of the command sequence in the latent space. It is then fed as input to a fully connected layer with weights $\mathbf{W}_{fc}$, followed by a softmax layer [6] for predicting the succeeding data command in the given sequence.

$$x_i = \mathbf{W}_e c_i,\ i \in [0, L] \tag{5}$$

$$v_i = LSTM(x_i),\ i \in [0, L] \tag{6}$$

$$P = \sigma(\mathbf{W}_{fc} v_L + b_{fc}) \tag{7}$$

$$Pr(\hat{dc} = dc_i) = \frac{exp(P[i])}{\sum_{i=0}^{N} exp(P[i])} \tag{8}$$

where N is the number of commands in DC, $b_{fc} \in \mathbb{R}$ is the bias term, P is a vector of size equal to N, $P[i]$ is the i^{th} value of the vector P, $\hat{dc} \in DC$ is the immediate data command succeeding the command sequence S and $dc_i \in$ DC is the i^{th} data command. Here x_i is a k-dimensional vector corresponding to the i^{th} command in the input sequence and σ is the standard sigmoid non-linearity [31] applied to the output of the fully connected layer. Mathematically,

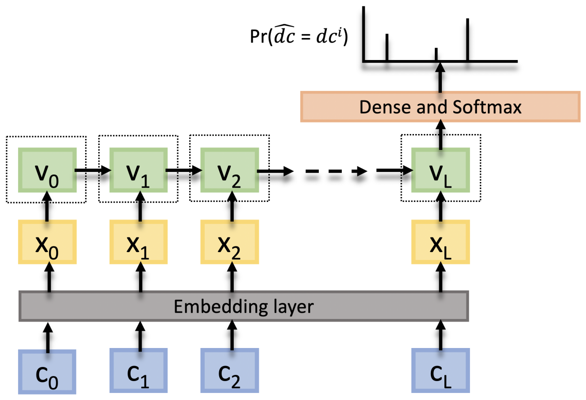


Figure 3: The proposed LSTM based architecture diagram for predicting the next data command in the sequence. This vanilla architecture is unaware of intent information.

at each step of data command prediction, the following probabilities are computed to generate next data command in the sequence S:

$$Pr(\hat{dc} = dc_i | c_0, c_1, \ldots, c_L) \tag{9}$$

K such models are trained, one for each of the K intents identified through the BTM model. These models differ in terms of the input data provided, weight matrices $\mathbf{W}_e$, $\mathbf{W}_{fc}$ and bias term b_{fc}. The output data command space, DC, is same for all the K models. The commands which are model predictions are the ones which have the top probabilities.

$$\arg\max_{dc_i \in \text{DC}} Pr(\hat{dc} = dc_i | c_0, c_1, \ldots, c_L) \tag{10}$$

Gehring et. al [14] have shown the tantamount performance of convolutional neural networks (CNN) compared to recurrent neural networks (RNN) in sequence modeling. Compared to recurrent models, computations over all elements can be fully parallelized during training facilitating us to better exploit the GPU hardware and optimization. This is due to two reasons (1) the fixed number of non-linearities in the model architecture and (2) the independence between the number of non-linearities and the length of input command sequence. Consequently, we have also explored the space of CNN's in modeling the command sequences.

We now turn to describe the CNN model. We use the standard notations defined in [19] to describe the model. The equations 5, 7 and 8 are used for embedding the commands and predicting the next data command in the given input command sequence. The equation 6, which is used for computing the representation of the command sequence $(c_0, c_1, \ldots, c_L)$, is modified as follows. The embedded command sequence of length L (padded where necessary) is represented as

$$x_{0:L} = x_0 \oplus x_1 \oplus x_2 \oplus \cdots \oplus x_L \tag{11}$$

where $\oplus$ is the concatenation operator. In general, let $x_{i:i+j}$ refer to the concatenation of embedded commands $x_i, x_{i+1}, \ldots x_{i+j}$. A convolution operation involves a filter $\mathbf{W}_f \in \mathbb{R}^{h \times k}$, which is applied to a window of h embedded commands to produce a new feature. The feature f_i is thus generated from a window of embedded commands $x_{i:i+h-1}$ by the following equation.

$$f_i = \sigma(\mathbf{W}_f \cdot x_{i:i+h-1} + b_f) \tag{12}$$

Here $b_f \in \mathbb{R}$ is the bias term. The convolutional filter is applied to each possible location in the command sequence. Therefore, the following feature map is generated by applying the filter to the possible windows $\{x_{0:h-1}, x_{1:h}, \ldots, x_{L-h+1:L}\}$

$$\mathbf{f} = [f_0, f_1, f_2, \ldots, f_{L-h+1}] \tag{13}$$

with $\mathbf{f} \in \mathbb{R}^{L-h+1}$. This feature vector is provided as input to a max-over-time pooling operation layer [9]. In this operation, the maximum value of the vector, $\hat{f}_{cf_1} = max\{\mathbf{f}\}$, is produced as output for that particular convolutional filter represented by cf_1. The motivation behind this operation is to capture the most significant feature—one with the highest value—for each feature map. The max-over-time pooling operation also handles the problem of having variable length command sequences. The features $\hat{f}_{cf_i}$, thus obtained from the i^{th} convolutional filter, are concatenated to produce a single representation of the command sequence. This representation, let be v_L, is further processed, as mentioned in equations 7 and 8, for predicting the next data command in the given sequence. The mathematical representation of this model remains the same as shown in equation 9.

4.4 Intent Informed Models

In this section, we propose models that are provided with intent information explicitly. While ensemble of classifiers method is a competitive baseline for multi-class classification task, this approach of providing the task information implicitly fails in our setting. These ensemble of models lack generalizability, while predicting data commands for sequences which do not align with the intent. Consequently, we impose additional constraint on the seq2seq model to capture the relation between intent and the command sequence by providing the information explicitly [21, 29]. This approach has the advantage of training the model on relatively larger set of data thereby resulting in better accuracy [3]. We experiment with three ways of incorporating the intent information into the models each of which perform in par with the other two models.

4.4.1 ***Intent Concatenated Representation*** *(InCoRe).* While the intent information can be provided to the traditional LSTM models in several ways, this model chooses to administer this information at the fully connected layer. Providing the intent information at the fully connected layer aids the model to keep the track of intent while predicting the next data command. The one hot representation of the intent is concatenated to the effective representation of the input command sequence, obtained after the LSTM encoder layer. The reasons why we chose one hot representation for the intents are two-folded. (1) It is desired that the intents should have distinguishable representations, and one-hot representation distinguishes each intent in the intent vocabulary from every other intent. (2) The intents should not be prioritized in terms of representations when a global model is trained on data points from all the intents. Accordingly, the equation 7 is modified to:

$$P = \sigma(\mathbf{W}'_{fc}[v_L \oplus 1_I] + b'_{fc}) \tag{14}$$

Here 1_I is the one hot representation of the intent, I, assigned to the command sequence $(c_0, c_1, \ldots, c_L)$ by the BTM. The equations 5, 6 and 8 remain the same for this model whereas the equation 9 is altered to:

$$Pr(\hat{dc} = dc_i | c_0, c_1, \ldots, c_L, I) \tag{15}$$

Here, I is the intent assigned to the command sequence $(c_0, c_1, \ldots, c_L)$ by the BTM. The modified equations can be applied to the CNN model as well, with the rest of the equations remaining intact.

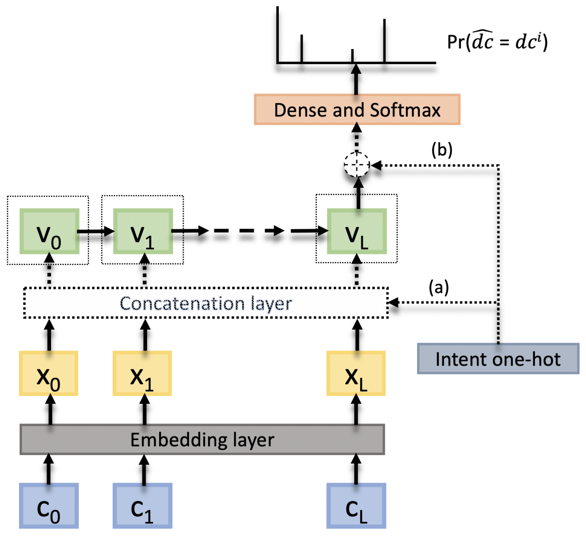


Figure 4: The architecture diagram for intent informed (a) *InCoRe* and (b) *InComm* models.

4.4.2 ***Intent Concatenated Commands*** *(InComm).* In the previous approach, the encoded representation after the LSTM layer does not take the intent into account. As a result, it will be hard to find distinct intent clusters in the latent representation space. Therefore, in this approach and the subsequent approach, we try to provide the intent information before the LSTM layer. It is expected from these models to incorporate the intent information while representing the input sequences and thereby provide better results. The one hot representation of the intent is concatenated to each of the embeddings of the commands in the sequence and is fed as input to the LSTM encoder. Accordingly, the modified equations are:

$$d_i = x_i \oplus 1_I,\ i \in [0, L] \tag{16}$$

$$v_i = LSTM(d_i),\ i \in [0, L] \tag{17}$$

The equations 5, 7 and 8 remain the same for this model. Correspondingly, the equation for the encoding part of the CNN model is:

$$f_i = \sigma(\mathbf{W}_f \cdot d_{i:i+h-1} + b_f) \tag{18}$$

4.4.3 ***Intent Appended Inputs*** *(InAI).* This model assumes the intents and the commands to be in the same latent representation space [13]. The intent information is provided as input at the first time step of the LSTM unit similar to the proposed architecture in [43]. The intent and command embeddings are trained together in the same k-dimensional space to generate cluster representative embeddings for the intents. Each intent is assigned a fixed index ranging from 1 to K with offset equal to N. The embedding matrix $\mathbf{W}_e$ is modified to facilitate the embeddings for the intents. The intent is prepended to command sequence and is fed as input to the model. The modified input sequence to the LSTM will be $(I, c_0, c_1, \ldots, c_L)$. Consequently, the equations 5, 6 and 7 are modified to:

$$x_i = \begin{cases} \mathbf{W}_e^{'} I & i = 0 \\ \mathbf{W}_e^{'}[c_{i-1}] & i \in [1, L+1] \end{cases} \tag{19}$$

$$v_i = LSTM(x_i),\ i \in [0, L+1] \tag{20}$$

$$P = \sigma(\mathbf{W}_{fc} v_{L+1} + b_{fc}) \tag{21}$$

The equation 8 remains the same. The mathematical representation of the model is given by equation 15.

4.5 Loss Function

So far the various proposed models are variations of next data command prediction models based on seq2seq modeling. They differ in how we give input sequences from different intents, and the way the intent is provided to these models as described in sections 4.3 and 4.4. Some of these models have been trained on sequences from specific intents, while others have been explicitly provided the intent as input. Therefore, it is expected from the models to implicitly learn the relation between the intents and their corresponding command distributions. The standard cross-entropy loss function [17], when applied to these models, makes sure that the recommended data commands are actually aligned with the input sequence.

$$\mathcal{L}_{CE}(\theta) = \frac{1}{M} \sum_{n=1}^{M} - \log Pr(\hat{dc} = dc_m | c_{0_m}, c_{1_m}, \ldots, c_{L_m}, I_m) \tag{22}$$

where M is the total number of sequences and dc_m is the ground truth output for the m^{th} input command sequence (c_{0_m}, c_{1_m}, ..., c_{L_m}), I_m is the assigned intent. However, a severe limitation to this standard loss function in our setting is that this loss function does not consider the intent orientation while penalizing the models. These models do not have access to crucial information provided by the BTM, which is the definition of an intent.

For this, we use the definition of intents, the probability distribution over data commands $P(dc|intent)$, and incorporate it into the loss function. In order to force this, we introduce the Kullback-Leibler divergence [20] into the loss function. The KL divergence measures how much a probability distribution P differs from a second distribution Q. Minimizing the KL divergence means optimizing the probability distribution $Pr(\hat{dc} = dc_m | c_{0_m}, c_{1_m}, \ldots, c_{L_m}, I_m)$ to be close to the data command distribution of the intent $P(dc|intent)$.

The distributions of interest in computing KL divergence are as follows

$$\mathcal{L}_{\text{KL}}(\theta) = \mathcal{D}_{\text{KL}}(P||Q) \tag{23}$$

where $P = Pr(\hat{dc} = dc_m | c_{0_m}, c_{1_m}, \ldots, c_{L_m}, I_m)$ and $Q = P(dc|intent)$. We want our model to give recommendations that have a high $P(dc|intent)$ when the input from the user deviates from the chosen intent. Cross entropy loss penalizes prediction deviating from the ground truth command. To penalize deviation from the intent, we introduce a loss function, $\mathcal{L}(\theta)$, that takes care of both the penalties:

$$\mathcal{L}(\theta) = \alpha \mathcal{L}_{\text{CE}}(\theta) + (1 - \alpha)\mathcal{L}_{\text{KL}}(\theta) \tag{24}$$

The first term is a scaled version of the cross entropy loss. The second term is the KL Divergence between the predicted probability distribution and the chosen intent distribution. α is the balancing factor for both the losses.

4.6 Fine Tuning

The loss function described in equation 24, when applied to the models proposed in Section 4.4, steers the recommender systems to produce data commands relevant to the intent at hand. The proposed loss function involves the component Q, which is different for different intents. Therefore, it is not possible to train a single global model for all the intents using this loss function. A simple solution is to train K different models for K different intents, similar to what we proposed in Section 4.3. These K different models are trained on K intent specific datasets with loss function $\mathcal{L}(\theta)$. The models used in this experiment are described in Section 4.4. Such models are only exposed to the sequences specific to an intent and thus display aberrant behavior for previously unseen command sequences from other intents resulting in loss of generalizability. The recommendations might not be diversifying as well, in representing the sequences that are provided as input to the model. Moreover the intent specific data models yield aberrant outputs for adversarial inputs, where the user deviates from the specified intent. In order to assuage these limitations, we adopt the techniques of transfer learning, described in [32] and fine-tuning, described in [34, 35].

Models trained on large volume of diversifying data from different intents overcome the problem of overfitting [40]. For this reason, the intent informed models, when trained on global data using the loss function $\mathcal{L}_{\text{CE}}(\theta)$, have an inherent advantage of learning to represent the sequences better compared to the intent specific data models. These representations are captured through the embeddings $\mathbf{W}_e$, the hidden state weights of the LSTM layer, and the fully connected layer parameters, $\mathbf{W}_{fc}$ and b_{fc} of these models. Therefore we utilize the trained weights of these models for initializing the parameters of this experiment. We then fine tune the models to be intent specific by retraining the models on the data specific to this intent with the modified loss function $\mathcal{L}(\theta)$. Consequently, these fine tuned models are able to provide accurate and intent relevant data command recommendations. Another indispensable advantage of these models is their superior performance for low resource intents, that is, intents with less training data.

5 EXPERIMENTS

5.1 Baseline Approaches

Limited research has been carried out in identifying and incorporating intent information for predicting next action in a sequence. Therefore, we compare our proposed models with simple, yet powerful, approaches used for next action prediction of which are the popularity based prediction models and Markov models.

5.1.1 ***Random Model***.
Given an input command sequence S, we randomly predict one among the 766 data commands to be output. The output of this model is a randomly generated probability distribution over the data commands. Following this distribution, a data command is chosen. The odds for correctly predicting the next data command is 1 to 765.

5.1.2 ***Frequency Model***.
In this model, we assign the probability of generating a data command to be equal to the probability of finding the data command in the dataset. The probability distribution, thus generated is observed to follow Zipf's law [49]. Few data commands have a high probability of being observed in the dataset. With this observation, we modified this baseline approach to predict only the 50 most frequent data commands. These 50 data commands constitute more than 90% occurrence of the total number of data commands in the dataset. The corresponding probability distribution is generated by normalizing over these top 50 data commands to a unit vector.

5.1.3 ***Markov Model***.
The prominent approach with historical information of commands is the Markov models. The N^{th} order Markov model depends on the N previous commands to predict the next command. We compare our models with simple, yet powerful and promising baseline, the First-order Markov model. Since there are two sets of commands and only one set of commands is being considered for prediction, appropriate modifications were introduced into this approach. The traditional formula for computing the probability of observing a data command, dc_i, given the most recent command c_L is:

$$Pr(\hat{dc} = dc_i | c_L) = \frac{count(c_L, dc_i)}{count(c_L)} \tag{25}$$

where $count(c_L)$ is the frequency of command c_L in the dataset and $count(c_L, dc_i)$ equals the frequency of the *biterm* (c_L, dc_i). It is to be noted that the term *biterm* is used instead of *bigram* in standard Markov model. Generating biterms from the dataset is explained through the following example. For a given sequence $(dc_1, dc_2, sc_1, sc_2, dc_3, sc_3, dc_4)$, the biterm set is $\{(dc_1, dc_2), (dc_2, dc_3), (sc_1, dc_3), (sc_2, dc_3), (dc_3, dc_4), (sc_3, dc_4)\}$, where $sc_i \in$ SC and $dc_i \in$ DC. In generating the biterms for the first-order Markov model, the second term in the biterm is always from the set DC, whereas the first term can be from either command sets.

5.1.4 ***Vanilla model***.
The architecture of vanilla model is similar to the proposed intent informed models in Section 4.4 with no intent information being provided. It is a deep neural seq2seq model trained on entire data to predict the next data command in the sequence.

5.2 Intent Identification

The first task in identifying the intents from the dataset is to choose the optimal number of intents for the dataset at hand. We have utilized the measures described in Section 4.2 to decide upon this number. We computed intent coherence scores $CS_{UCI}(t)$ and $CS_{UMass}(t)$ by iterating the number of intents from $t = 1$ through 50 to determine the optimal number of intents. Through a comparative analysis across these scores for various values of t, $t = 6$ had the highest average coherence score.

For intent identification using the BTM, standard Gibbs sampling [47] is used to compute the values of the multinomial distributions. The choice of the hyperparameters α and β is based upon the desired output distribution of BTM. The value of α is set to 8.333 and β is set to 0.005.

5.3 Model configurations

For the purpose of our proposed LSTM-based models, we consider the sequence $(c_0, c_1, \ldots, c_L)$, S, to be the input along with the assigned intent I. Here the length of S is chosen to be 30. This is the average value of the length of command sequences in the dataset. The output to be predicted for each of these input command sequences is the immediate data command after the sequence as described in Figure 1. The ground truths are obtained as described in Section 3. Each command c_i is represented as a 200 dimensional vector in the embedding space. These embeddings, when trained along with the models, learn to capture the semantic meaning of the commands in the setting of log data. The user is asked to choose one of the intents from the list of the identified intents from the dataset at the start of each session. Each intent is indexed from 1 to K, where K is set to 6 for our dataset. The intent is represented as the same 200 dimensional vector for *InAI* unlike the *InComm* and *InCoRe* models, where it is represented as a one-hot vector. At each time step T, the embedded command x_i is either concatenated with the intent's one hot representation, for *InComm* model, or is directly provided as input to the LSTM layer, for *InAI* and *InCoRe* models. The number of hidden LSTM units is set to 500 to generate the representation of the input sequence, followed by a fully connected layer. The input to the fully connected layer is a 500 dimensional representation of the input command sequence, v_L, for the *InComm, InAI* models and 506 dimensional intent concatenated representation for the *InCoRe* model. We further utilize dropout probability of 0.5 in the dense layer. During training, we minimize the cross-entropy loss $\mathcal{L}_{\text{CE}}(\theta)$ between the logits, the predicted probability distribution over the entire data commands, and the ground truth distribution. The optimizer used is a stochastic gradient descent (SGD) with 0.01 learning rate, 0.9 momentum and weight decay of 0.001 to train our models. The weights of these models are initialized randomly with no prior distribution.

For the fine tuning experiments, the weights are initialized to the trained values of *InCoRe, InComm, InAI* models. During fine tuning, we set the embeddings of the commands to be fixed. For each of the three models, *InCoRe, InComm*, and *InAI* models, 6 models are fine-tuned taking each of the existing trained models as the base models. While fine-tuning, the cross-entropy loss function, $\mathcal{L}_{CE}(\theta)$ is modified to $\mathcal{L}(\theta)$. The value of α in the equation 24 is set to 0.5 based on empirical observations.

6 EVALUATION

6.1 Intent Coherence

We recall that each intent is a probability distribution over the same set of data commands. To evaluate the quality and coherence of identified intents by BTM, which was trained in an unsupervised fashion, we rely on the assessment of two experts who have several years of experience with the software under consideration. These experts were shown top 50 commands for each of the 6 intents. They were able to relate to each collection of commands and provide a phrase describing each intent. These descriptions read as follows: product analysis for campaign or marketing, search trends for campaigning research, product conversions on different devices, navigational research and funnel analysis, segment analysis for campaigning research, and search trends for web-product analysis.

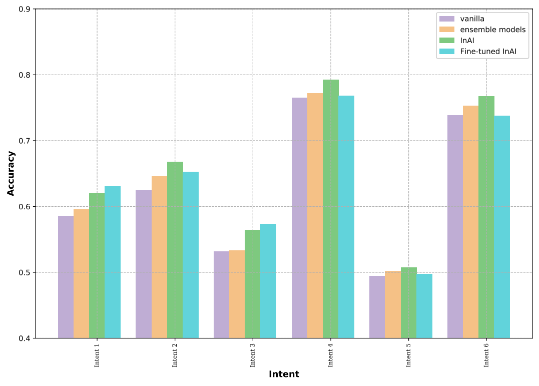


Figure 5: Accuracy values of proposed neural architecture models for each intent. The reported values are for the LSTM version of the models. *InAI* model outperforms for 4 out of the 6 identified intents.

6.2 Accuracy of Models

To quantify the performance of our models, we computed the standard evaluation metric, the test accuracy. In our setting, the test accuracy is defined as the number of examples where the predicted data command matches with the target data command out of the total number of examples. This is done at the granularity of each command in the ground truth. For uniformity, the test set across all the models was exactly the same.

From the accuracy values, it is empirically observed that the models incorporating the intent information, in general, perform better (by at least 19% margin) compared to the ones that do not. Intent information influences the process of data command recommendation to make it more germane which is clearly reflected from the results reported in Table 1. The three Intent Informed models introduced in Section 4.4, *InCoRe*, *InComm* and *InAI*, achieve similar accuracy values. The *InAI* model performed best among the three models as showcased in Table 1. The intent-wise accuracy results are plotted in Figure 5. The fine-tuned models also have obtained similar accuracy values with Fine-tuned *InAI* having best performance as observed from Table 2.

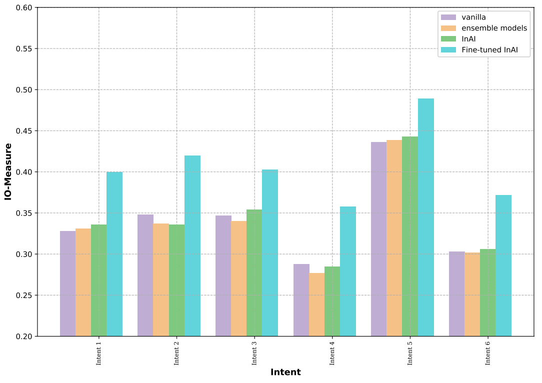


Figure 6: Intent orientation IO_1 scores for the proposed neural architecture models for each intent. The reported values are for the LSTM version of the models. The fine-tuned *InAI* model has a significant improvement in IO_1 score compared to other models.

Model		Accuracy	IO_1 Score
Random Model		0.0614	0.0808
Frequency model		0.2520	0.1428
Top 50 Frequency		0.3633	0.1997
First-order MM		0.4521	0.2251
Second-order MM		0.4710	0.2322
vanilla	LSTM	0.5875	0.3776
	CNN	0.5230	0.4011
ensemble models	LSTM	0.6894	0.3211
	CNN	0.5023	0.3724
InCoRe	LSTM	0.6839	0.3984
	CNN	0.5258	0.4065
InComm	LSTM	0.6970	0.3959
	CNN	0.5478	0.4254
InAI	LSTM	0.7189	0.3933
	CNN	0.5306	0.4396

Table 1: Accuracy results and Intent Orientation IO_1 scores of various models, with best results in gray background.

6.3 Intent Orientation Measure (*IO-Measure*)

Due to the very nature of the problem we attempt to solve, conventional evaluation metrics such as test accuracy will not evaluate the effectiveness of our model, as to what extent are we able to solve the problem. Accuracy is the measure of the number of input sequences for which our model predicts the next data command taken by the user. It does not convey how well the recommendations were aligned with the selected intent. If we assume that the sequence of commands are completely coherent with the chosen intent, and so is the labeled next data command, only then will accuracy be reliable.

Also, in a practical situation the user might deviate from the chosen intent, in which case our model should give more weight to the chosen intent in order to direct the user towards the same through recommendations, and less importance to the most recent sequence of actions taken by the user. This is tested by giving the model sequences from all intents but the chosen intent and analyzing the recommendations. Here as well, plain accuracy measure will fail to give a accurate measure of recommendations' effectiveness.

Model		Accuracy	IO_1 Score
vanilla	LSTM	0.5563	0.3852
	CNN	0.5052	0.4132
Fine-tuned *InCoRe*	LSTM	0.6647	0.4322
	CNN	0.5011	0.3833
Fine-tuned *InComm*	LSTM	0.6820	0.4268
	CNN	0.5366	0.4421
Fine-tuned *InAI*	LSTM	0.7010	0.4811
	CNN	0.5300	0.4810

Table 2: Accuracy results and Intent Orientation IO_1 scores of fine-tuned models with and without intent information. The fine-tuning approach is always beneficial.

Model		Accuracy	IO_1 Score
ensemble models	LSTM	0.1525	0.2566
	CNN	0.1010	0.3022
InAI	LSTM	0.4919	0.1966
	CNN	0.4516	0.2355
Fine-tuned *InAI*	LSTM	0.2795	0.4823
	CNN	0.2231	0.4763

Table 3: Accuracy results and Intent Orientation IO_1 scores of the proposed models when tested on adversarial examples. The proposed IO_1 measure decreases sharply for the *InAI* model when trained on $\mathcal{L}_{CE}(\theta)$ loss function compared to the fine-tuned *InAI* model, which is trained on $\mathcal{L}(\theta)$ loss.

Thus, to bring in the notion of intent awareness in the evaluation as well, we bring in the intent awareness score: the probability of the recommendation given the intent $P(dc|intent)$. Since the objective is to evaluate the model both in terms of intent awareness score and accuracy, we utilize a measure similar in spirit to the F-Measure [38], which we term as the *IO-measure* and is defined as:

$$IO_\beta = (1+\beta^2) \cdot \frac{accuracy \cdot \overline{P}(dc|\text{intent})}{(\beta^2 \cdot accuracy) + \overline{P}(dc|\text{intent})} \tag{26}$$

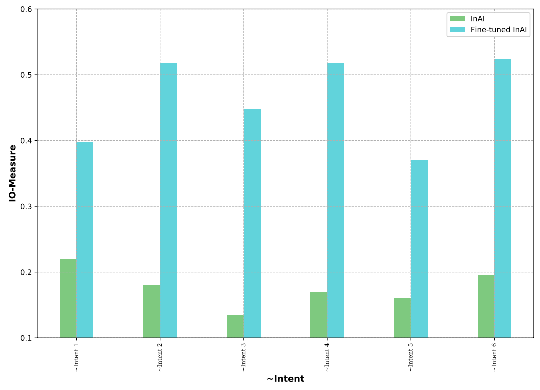
where $\overline{P}(dc|intent)$ is the average value of $P(dc|intent)$ for the predicted data command dc across the test examples for a specific intent. The value of β measures the effectiveness of predicting commands with respect to a session which attaches β times as much importance to accuracy as intent orientation [42]. This is a balance between personalization and intent orientation. We present results of our models for this measure with $\beta = 1$ score. The value of β can be varied depending on the requirement from the use-case.

The IO_1 scores of the proposed models and fine-tuned models are reported in Tables 1 and 2 respectively, with intent-wise results plotted in Figure 6. The *InAI* model performs better compared to other models when trained on loss functions $\mathcal{L}_{CE}(\theta)$ as well as $\mathcal{L}(\theta)$. The improved values of IO_1 reported in Table 2, after fine-tuning the models, exhibit the recommended data commands being intent aligned with accuracy being intact. Results point that we can obtain significant improvements in terms of IO_1 through fine-tuning, i.e., at least 12.7% and at most 24.9% (for LSTM) indicating that this is very a promising direction to pursue further investigations.

6.4 Adversarial testing

The motivation behind introducing the fine-tuned models with custom loss function $\mathcal{L}(\theta)$ is to deliver intent oriented recommendations to the user. Models trained on data specific distribution often display aberrant behavior when tested on an example from different data distribution. It is often possible, in a real case scenario, that the user might deviate from the specified intent while progressing the session. In such cases, it is expected from the fine-tuned models to recommend data commands related to the specified intent and bring back the user on track. In this section, we test our models in such cases and display the robustness of the fine-tuned models.

For this experiment, the model is provided inputs from data distributions different from what it was trained on. For each intent, the specific fine-tuned model, datapoints corresponding to the other intents are provided as input. The intent-wise results are shown in



Note: The ~ Intent i on x-axis implies the model fine-tuned for intent i and tested on data from rest of the intents.

Figure 7: Intent-wise Intent Orientation IO_1 scores of *InAI* and fine-tuned *InAI* models on adversarial examples. The fine-tuned *InAI* model significantly outperforms *InAI* for all the intents.

Figure 7 and the aggregated results in Table 3. Though the accuracy of the models has decreased, the intent awareness score is intact. This provides evidence that the model is attempting to recommend data commands that are more relevant to the intent at hand. The decrease in accuracy can be interpreted as follows. The fine-tuned models were trained to predict the accurate data command for a given sequence, and then were altered to tune the recommendations towards a specific intent. When the user deviates from the specified intent, the sequence of commands might not map to that intent. Such an example, when visualized in the latent representation space of the sequences, lies outside the intent clusters. Due to this reason, recommendations provided might not align with the original input sequence. However, the suggested data command steers the user back towards the intended task.

7 CONCLUSION AND FUTURE WORK

In this paper, we have investigated the effectiveness of incorporating intent information while recommending data commands to the user. The results of our intent-aware data command recommendation models, when compared to traditional intent-agnostic recommendation models, are quite promising and call for further explorations along this line of work. To the best of our knowledge, this is the first work which applies the fine-tuning paradigm to tailor the data commands recommendations towards the specified intent. We validate our models on a novel evaluation measure that balances both accuracy and degree of intent orientation in the provided data command recommendations.

In future work, we will experiment with more sophisticated models that can predict the user's intent in real time based on the progress of the session, which currently is an input signal given by user at the start of the session. We believe that this line of work can be extended to handle the problem of a novice user misspecifying intent. With the promise of the fine-tuning approach, we would like to explore the recent advances in transfer learning to further boost the performance of our models.

REFERENCES

[1] Sara Alspaugh, Bei Di Chen, Jessica Lin, Archana Ganapathi, Marti A Hearst, and Randy H Katz. 2014. Analyzing Log Analysis: An Empirical Study of User Log Mining.. In *LISA*. 53–68.

[2] Loulwah AlSumait, Daniel Barbará, James Gentle, and Carlotta Domeniconi. 2009. Topic significance ranking of LDA generative models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 67–82.

[3] Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 26–33.

[4] Bibek Behera. 2016. Chappie-a semi-automatic intelligent chatbot. *Write-Up* (2016).

[5] Biswarup Bhattacharya, Iftikhar Burhanuddin, Abhilasha Sancheti, and Kushal Satya. 2017. Intent-Aware Contextual Recommendation System. In *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*. IEEE, 1–8.

[6] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. Springer. 115 pages.

[7] Ann Blandford. 2001. Intelligent interaction design: the role of human-computer interaction research in the design of intelligent systems. *Expert Systems* 18, 1 (2001), 3–18.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[9] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (almost) from Scratch. *J. Mach. Learn. Res.* 12 (2011), 2493–2537.

[10] Brian D Davison and Haym Hirsh. 1998. Predicting sequences of user actions. In *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*. 5–12.

[11] Himel Dev and Zhicheng Liu. 2017. Identifying frequent user tasks from application logs. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*. ACM, 263–273.

[12] Adji B. Dieng, Chong Wang, Jianfeng Gao, and John William Paisley. 2016. TopicRNN: A Recurrent Neural Network with Long-Range Semantic Dependency. *Computing Research Repository* (2016).

[13] Nemanja Djuric, Hao Wu, Vladan Radosavljevic, Mihajlo Grbovic, and Narayan Bhamidipati. 2015. Hierarchical neural language models for joint representation of streaming documents and their content. In *Proceedings of the 24th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 248–255.

[14] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 1243–1252.

[15] Nicolaus Henke, Jacques Bughin, Michael Chui, James Manyika, Tamim Saleh, Bill Wiseman, and Guru Sethupathy. 2016. The age of analytics: Competing in a data-driven world. *McKinsey Global Institute* 4 (2016).

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[17] Katarzyna Janocha and Wojciech Marian Czarnecki. 2017. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659* (2017).

[18] Bernard J. Jansen, Danielle L. Booth, and Amanda Spink. 2007. Determining the User Intent of Web Search Engine Queries. In *Proceedings of the 16th International Conference on World Wide Web (WWW '07)*. ACM, New York, NY, USA, 1149–1150. https://doi.org/10.1145/1242572.1242739

[19] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[20] Solomon Kullback. 1997. *Information theory and statistics*. Courier Corporation.

[21] Jey Han Lau, Timothy Baldwin, and Trevor Cohn. 2017. Topically Driven Neural Language Model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 355–365.

[22] Xumin Liu. 2013. Incorporating User Behavior Patterns to Discover Workflow Models from Event Logs. *International Conference on Web Services* (2013), 171–178.

[23] Xumin Liu. 2014. Unraveling and learning workflow models from interleaved event logs. In *2014 IEEE International Conference on Web Services*. IEEE, 193–200.

[24] Yi Liu and Yuan F Zheng. 2005. One-against-all multi-class SVM classification using reliability measures. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 2. IEEE, 849–854.

[25] Eddy Mayoraz and Ethem Alpaydin. 1999. Support vector machines for multi-class classification. In *International Work-Conference on Artificial Neural Networks*. Springer, 833–842.

[26] Rishabh Mehrotra, Mounia Lalmas, Doug Kenney, Thomas Lim-Meng, and Golli Hashemian. 2019. Jointly Leveraging Intent and Interaction Signals to Predict User Satisfaction with Slate Recommendations. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 1256–1267. https://doi.org/10.1145/3308558.3313613

[27] Tova Milo and Amit Somech. 2016. React: Context-sensitive recommendations for data analysis. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2137–2140.

[28] David Mimno, Hanna M Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. 2011. Optimizing semantic coherence in topic models. In *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 262–272.

[29] Aadhavan M Nambhi, Bhanu Prakash Reddy, Aarsh Prakash Agarwal, Gaurav Verma, Harvineet Singh, and Iftikhar Ahamath Burhanuddin. 2019. Stuck? No worries!: Task-aware Command Recommendation and Proactive Help for Analysts. In *Proceedings of the 27th ACM Conference on User Modeling, Adaptation and Personalization*. ACM, 271–275.

[30] David Newman, Youn Noh, Edmund Talley, Sarvnaz Karimi, and Timothy Baldwin. 2010. Evaluating topic models for digital libraries. In *Proceedings of the 10th annual joint conference on Digital libraries*. ACM, 215–224.

[31] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378* (2018).

[32] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[33] Kemal Polat and Salih Güneş. 2009. A novel hybrid intelligent method based on C4. 5 decision tree classifier and one-against-all approach for multi-class classification problems. *Expert Systems with Applications* 36, 2 (2009), 1587–1592.

[34] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf* (2018).

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019).

[36] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–59.

[37] Ryan M. Rifkin and Aldebaro Klautau. 2004. In Defense of One-Vs-All Classification. *J. Mach. Learn. Res.* 5 (2004), 101–141.

[38] Yutaka Sasaki et al. 2007. The truth of the F-measure. *Teach Tutor mater* 1, 5 (2007), 1–5.

[39] Belle Selene Xia and Peng Gong. 2014. Review of business intelligence through data analysis. *Benchmarking: An International Journal* 21, 2 (2014), 300–311.

[40] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1701–1708.

[41] Zhiqiang Tao, Sheng Li, Zhaowen Wang, Chen Fang, Longqi Yang, Handong Zhao, and Yun Fu. 2019. Log2Intent: Towards Interpretable User Modeling via Recurrent Semantics Memory Unit. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*. ACM, New York, NY, USA, 1055–1063. https://doi.org/10.1145/3292500.3330889

[42] C. J. van Rijsbergen. 1979. Information Retrieval. In *Encyclopedia of GIS*.

[43] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3156–3164.

[44] Michalis Vrigkas, Christophoros Nikou, and Ioannis A Kakadiaris. 2015. A review of human activity recognition methods. *Frontiers in Robotics and AI* 2 (2015), 28.

[45] Xu Wang, Benjamin Lafreniere, and Tovi Grossman. 2018. Leveraging community-generated videos and command logs to classify and recommend software workflows. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 285.

[46] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2013. A Biterm Topic Model for Short Texts. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13)*. 1445–1456.

[47] Ilker Yildirim. 2012. Bayesian inference: Gibbs sampling. *Technical Note, University of Rochester* (2012).

[48] Yuchen Zhang, Weizhu Chen, Dong Wang, and Qiang Yang. 2011. User-click modeling for understanding and predicting search-behavior. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1388–1396.

[49] George Kingsley Zipf. 1949. Human behavior and the principle of least effort. (1949).