WEAKLY SUPERVISED USER INTENT DETECTION FOR MULTI-DOMAIN DIALOGUES

Ming Sun[†] *Aasish Pappu*[§] *Yun-Nung Chen*[†⋆] *Alexander I. Rudnicky*[†]

[†]Carnegie Mellon University, Pittsburgh, PA, USA
[§]Yahoo Research, New York, NY, USA
[⋆]National Taiwan University, Taipei, Taiwan

ABSTRACT

Users interact with mobile apps with certain intents such as finding a restaurant. Some intents and their corresponding activities are complex and may involve multiple apps; for example, a restaurant app, a messenger app and a calendar app may be needed to *plan a dinner with friends*. However, activities may be quite personal and third-party developers would not be building apps to specifically handle complex intents (e.g., a DinnerPlanner). Instead we want our intelligent agent to actively learn to understand these intents and provide assistance when needed. This paper proposes a framework to enable the agent to learn an inventory of intents from a small set of task-oriented user utterances. The experiments show that on previously unseen user activities, the agent is able to reliably recognize user intents using graph-based semi-supervised learning methods. The dataset, models, and the system outputs are available to research community.

Index Terms— Multi-domain dialogue, user intent detection, mobile applications

1. INTRODUCTION

People interact with smartphone applications (apps) in pursuit of various goals (or intents). Some of them, such as finding a restaurant or navigating to a place can be fulfilled by single apps (e.g., YELP, MAPS). However, other intents may be complex and can involve using multiple apps together. For example, to plan an evening out or to schedule a group study with classmates, a series of actions across several apps may be performed [1, 2]. Currently, people manually (or verbally) launch the next app(s) and mentally carry over the context from previous app(s). It is desirable for the agent to learn complex user intents from daily app behavior of a user. One obvious utility, among others, would be that the agent would know which app the user would like to use immediately followed by the current active application [3].

As a natural communication modality, speech conveys user intents. For example, a user can express relatively simple intents such as *call Mom at home* or more complex ones like *I want to have a dinner with my family this Friday night*. To carry out a user intent, an agent should be able to map the content of a spoken input into sequence of actions. Our long-term goal is to let the agent automatically associate given spoken input with sequence of actions with few manually labeled intents. As a first step, an agent should automatically learn and maintain an inventory of complex user intents [4, 5]. The agent should add new intents to the inventory as it observes user's daily activities and also update existing intents based on user's feedback.

Having acquired an intent inventory, the next step is to recognize the user intent in current activity and update the inventory. Intent can be determined from two sources of information: app sequence and user's speech input. For both modalities, the input could be an ongoing activity, e.g., currently observed (unfinished) app sequence or simple speech commands. On the other hand, completed activities (e.g., a sequence of apps) should be either mapped to one of the known intents or recognized as an outlier. In this work, we investigate semi-supervised way to recognize complex user intent from a completed app sequence. The contributions of this paper are two-fold: The system 1) learns an inventory of complex user intents from spoken utterances and automatically cluster the intents, and 2) accurately recognizes complex intents by observing sequence of applications using graph-based semi-supervised methods.

In the rest of this paper, we first introduce the approaches to building intent inventory in Section 2, followed by experiments and results in Section 3. Finally we make concluding remarks and future directions. Our dataset, models and system output are released[1].

2. METHODOLOGY

2.1. Intent Inventory Learning

An inventory of complex user intents along with examples is maintained by the agent (see Fig 1). When enough new observations are accumulated, the agent updates the inventory with the following outcomes: 1) new intents can be formed; 2) new examples can be added to existing intents. This can be achieved by automatically clustering interactions into groups.

[1]https://github.com/aasish/userIntentDataset

Table 1. R_2 examples

App Sequence	Intent Description
WECHAT; SMS; WECHAT	getting friends to gather for karaoke
SMS; BROWSER; SMS	viewing a link Brooke sent me
BROWSER; YOUTUBE; APPSTORE; IHEARTRADIO; APPSTORE	listening to and trying to buy a new song
CLEANMASTER; SPIDERMAN; FACEBOOK; SPIDERMAN	inviting nerds to spiderman
CHROME; STEELERSRADIO	searching stats on the Steelers/listening to the Steeler game on the radio

Table 2. A breakdown of three different resources

Resource	# seqs	# apps/seq	# unique apps
R_1	6248	3.9 ± 3.4	369
R_2	1089	3.4 ± 2.4	188
R_3	533	2.4 ± 1.0	129

2.1.1. Resources

In order to form an array of intents, we use the following resources to mine user intents:

- R_1: sequences of apps users interact with on their smart phones. This is noisy since there are background apps (e.g., music is playing all the time).
- R_2: user knowledge of the intent-embedded sequence structures and the nature of the intents. Fig 2 illustrates user annotation on R_1. Table 1 shows examples of clean app sequences and the intent descriptions.
- R_3: speech commands to perform actions that compose complex intents (e.g., "Find a birthday song in YouTube" $\rightarrow$ "Post the song on Carter's Facebook"). Depending on the availability of manual transcripts, one can use speech recognition hypotheses instead (R_3'). We used Google ASR's top-1 hypotheses with WER = 23%.

Twenty seven (27) participants (27.7 ± 9.9 years old) enrolled in this longitudinal study. However, 8 of them did not show up after the first visit. App level activities (i.e., app invocations) are logged together with contexts such as time and location on a daily basis from the Android phones of the rest 19 participants (29.3 ± 11.1 years old) to form R_1. A time-based segmentation divides a sequence into two if there is an inactivity for 3 min. We observed, in a pilot study, that 3 min was optimal, although the threshold may vary across users. Participants could easily delete sequences from the logging interface for privacy purpose. R_2 requires supervision from the same participants who produce the app sequences (see Fig 2). A further 5 participants dropped out before we started to collect R_3. From the remaining 14 participants (31.3 ± 12.4 years old) we sampled a portion of R_2 and let users re-enact the same activity but by talking to a Wizard-of-Oz speech system instead of GUI interface [6]. The relationship among these three resources is: $R_3 \subset R_2 \subset R_1$. A breakdown of these three resources is shown in Table 2.

2.1.2. Learning Intent Clusters

To cluster observed activities into groups (step 1 and 3 in Fig 1), a vector representation for each data point and a clustering algorithm are required. Sequences in R_2 can be represented as $\vec{S} = [\vec{A}, \vec{D}]$, a concatenation of (fixed-size) vector representations of app sequence ($\vec{A}$) and intent description ($\vec{D}$) respectively. Similarly, activities in R_3 can be represented in the same fashion, with $\vec{D}$ representing word sequence in an array of speech commands. Instead of sparse one-hot vector based on the vocabulary of apps and descriptions, we use continuous representation based on app-embedding and word-embedding. To represent app sequence $A = \{a_1, a_2, ..., a_n\}$ we normalize the aggregated embeddings of each app: $\vec{A} = (\sum_{i=1}^{n} \vec{a_i})/n$. Here, each $\vec{a_i}$ can be trained similar to word2vec [7], but instead on a corpus of app occurrence (i.e., R_1). We call this `app2vec`. Similar idea has been proposed in [8]. Fig 3 shows that apps with similar functionality are close to each other in this embedding space. To get $\vec{D}$, we can either aggregate word embeddings or directly train a `doc2vec` representation. In this work, we added up embeddings of content words in the description sentence and normalize it with the length of the description.

Commonly used clustering methods such as k-means or Latent Dirichlet Allocation (LDA) [9] suffer from the requirement of a pre-determined number of clusters k. On the other hand, methods such as mean-shift, affinity propagation or g-means can automatically optimize k, which is ideal for our application. In this paper, we used affinity propagation.

2.2. Recognizing Complex User Intent

Given a sequence of mobile applications $\{a_1, a_2, a_3 \dots a_n\}$, we want to predict user's intent $C \in \{C_1, C_2, C_3 \dots C_k\}$ determined in the clustering process (described in the previous section). As a result, the agent can test its hypothesis of the current user intent by verbally confirming with the user, only when the confidence is not high (step 2 in Fig 1). However, it is expensive to obtain labeled data that accurately associate app sequences with intents. Therefore, we perform semi-supervised learning to label unlabeled sequences.

Semi-supervised learning (SSL) exploit input distribution

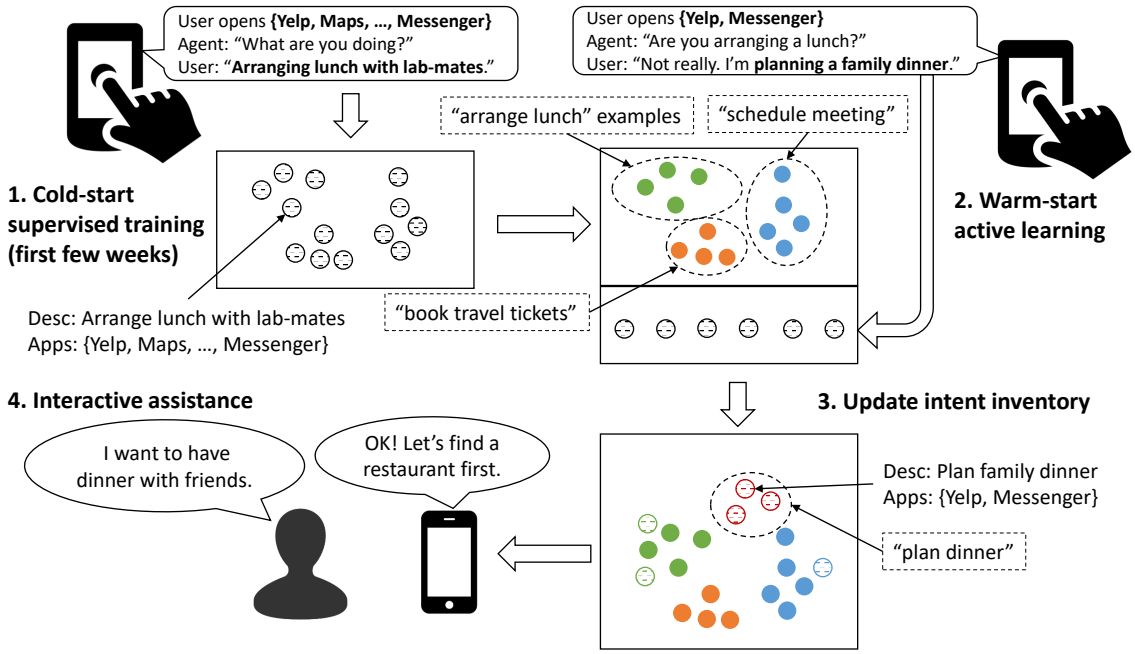

Fig. 1. Update intent inventory. Small circles denote individual multi-app interactions while dashed ones are newly observed and have not been clustered. Clusters of intents are automatically formed (coded with different colors). Part of the newly observed examples form new intent cluster(s); the rest add to existing intents.

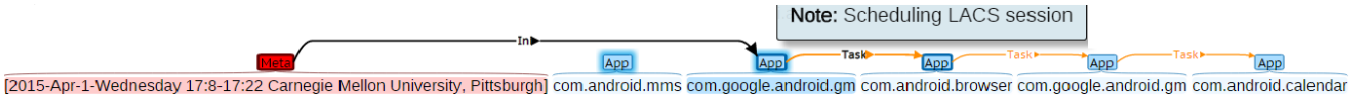

Fig. 2. Example of user annotation. User first connects apps that serve a common intent and then provides a brief description of the nature of the task. To aid recall, we provide context such as when and where this activity happened (if GPS available).

to learn a classification model. There are several SSL methods viz., generative models (Semisupervised Naive Bayes) [10], Transductive Support Vector Machines [11], Self-training methods that use kernel functions [12], entropy minimization [13] and graph-based methods [14, 15, 16, 17]. In this work we primarily focus on graph-based SSL methods that exploit latent relations between data points. Graph based methods are particularly known to work well in diverse tasks such as entity linking [18], sentiment lexicon induction [19], gloss finding [20] and other information extraction tasks. In this work, we study several graph construction methods and three graph-based inference methods. We discuss the problem formulation and the individual methods below. We construct an undirected graph from input data points, in our case every app sequence is a datapoint. We use both labeled and unlabeled sequences to construct this graph. The edge and its weight between a pair of datapoints can be determined using a distance metric, kernel function or a correlation function. Once the graph is constructed, seed labels are assigned to few nodes in the graph L and the labels are propagated to the unlabeled nodes U based on their affinity to the labeled nodes. The graph G is defined as (V, E, W), where $V \leftarrow L \cup U$, E edges, W similarity matrix for all nodes in V. The labeled nodes are assigned with seed-labels S and an iterative algorithm performs label assignment under the constraints of edge weights with hyperparameters that control the label propagation. In this work, we explored various graph construction techniques and graph-based SSL methods.

2.2.1. *Graph Construction*

We trained vector space representation for app sequences (`app2vec`) on the training data to obtain d dimensional vectors for every app a in our dictionary of apps A. Later we use this representation to infer a dense vector for every app sequence of variable length. Given n app sequences (both labeled and unlabeled), we use a vector representation for each one to construct a matrix of these vectors $X_{n \times d}$. To construct a graph from X, we explored several methods to build an adjacency matrix $W_{n \times n}$— 1) Nearest Neighbors 2) Covariance estimation 3) Kernel-based similarity.

Nearest Neighbors is a popular method for constructing graphs in graph-based SSL [21]. We used both KDTree

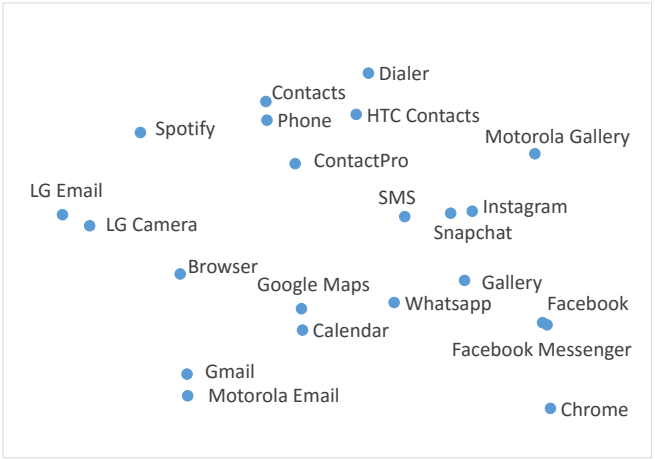

Fig. 3. Illustration for distributional app vectors. 30 dimension app vector is learned for each app in R_1 via word2vec. Frequently used apps are shown.

and BallTree algorithms [22] to construct the graphs. The BallTree algorithm constructs a binary tree in d dimensional space where every datapoint is treated as d dimensional hypersphere, in our case every app sequence is a hypersphere. The algorithm iteratively splits the datapoints based on their greatest separation in near-linear time ($O(nlogn)$). KD-tree operates in a similar fashion as BallTree.

Kernel-based Similarity We compute similarity matrix W given the matrix X (vertically stacked vectors of nodes) using different kernel functions such as:

1. linear $k(u, v) = uv^T$ which returns gram matrix
2. polynomial $k(u, v) = (\gamma u^T v + c_0)^x$
3. sigmoid $k(u, v) = tanh(\gamma u^T v + c_0)$
4. RBF $k(u, v) = \exp(-\gamma||u - v||^2)$
5. χ^2 kernel $k(u, v) = \exp(-\gamma \sum_i \frac{(u[i]+v[i])^2}{u[i]+v[i]})$

Covariance Estimation As an alternative to kernel estimation to construct the adjacency matrix, we could use estimated covariance matrix $\Sigma_{n\times n}$ of the label sequence matrix $X_{n\times d}$. We used several covariance estimation techniques listed below in our experiments:

1. *Empiricial Covariance*: This estimation technique computes maximum likelihood estimate of the labeled sequence matrix X. This technique is sensitive to outliers in the dataset, but serves as a good baseline.
2. *Shrunk Covariance*: Shrunk covariance is a transformation applied to the empirical covariance matrix where it reduces the ratio between the smallest and largest eigen value in the matrix. This essentially performs a convex transformation on the input covariance matrix, thus obtaining a robust estimate of the covariance matrix.
3. *Ledoit-Wolf* (LW): This technique performs a robust estimation of covariance matrix for large matrices [23]. It does not make assumptions about the distribution of the sample, thus can be applied to samples drawn from noisy data. The resultant covariance matrix is invertible and also well-formed. The method computes an asymptotically convex combination of the *Empirical* covariance and the identity matrix, results in robust covariance matrix.
4. *Pearson product-moment Correlation*: This technique computes pairwise Pearson correlation for all datapoints in X across all dimensions d. It returns a matrix $R_{n\times n}$ with values between -1 and 1.

2.2.2. *Graph based Labeling*

Given input matrix X, similarity matrix W and a set of labels C where only some of the datapoints in X are labeled L and rest are unlabeled U. We use the following methods to estimate label distribution for U.

Label Propagation algorithm [24] assumes a fully connected graph that uses a scaled version of euclidean distance to construct a weight matrix. The iterative algorithm allows the label distribution propagate to all nodes through edges. Edges with large weights allow labels to propagate easily and a probability transition matrix (between nodes) is estimated based on the edge weights. In every iteration all nodes propagate their label distribution to their neighbors until the distribution of every node convergences such that low-density regions in the input data are assigned with labels. The advantage of this algorithm is its simplicity but there's no theoretical convergence guarantee.

Adsorption is a transductive learning algorithm [25] that operates in noisy-label assumption and aims to relabel labeled examples for coherency across the graph. Adsorption performs a controlled random walk over the graph G and the control is determined by three hyperparameters `inject`, `continue`, and `abandon` and corresponding probabilities p^{inj}, p^{conj}, p^{abnd} for every node $v \in \{L \cup U\}$. Once the algorithm starts its random walk, with probability p^{inj}, it may stop and return the pre-defined label distribution Y. Alternatively, it can abandon the labeling and return all-zeros vector with probability p^{abnd}. Or it would continue the random walk from the current node to one of its neighbors with probability p^{cont}. Similar to label propagation, the transition probability between two nodes is a normalized edge weight between the nodes. The advantage of this algorithm is that it can be easily parallelized and scalable to large datasets. However, it has been shown that the minimization of Adsorption's objective function does not converge to local optima.

Modified Adsorption [26] improves on Adsorption by guaranteeing convergence to local optima. It differs from Adsorption in its objective function, which enforces the following conditions: (i) estimated labels for labeled nodes should

Table 3. First row shows graph statistics for the graph constructed through each process. Rows 2–4 show Kullback-Leibler Divergence between Task description Topic Distribution vs Semi-supervised multi-label multi-class topic distribution. Lower the Better. We see that only Kernel based methods perform the best among all graph-construction methods.

Graph	NN search		Kernel					Covariance			
	KD Tree	Ball Tree	linear	poly	sigmoid	χ^2	RBF	Emp	LW	Shrunk	Pearson
#Vertices *#Edges*	762 1714	842 2134	1087 90,919	1089 100,826	1087 105,938	1089 22,546	1089 135,975	1087 104,001	1087 104,759	1087 104,693	1087 252,325
Label Prop	6.974	4.095	0.226	0.099	0.100	0.138	**0.075**	0.100	0.100	0.100	0.129
Adsorption	6.917	3.982	0.253	0.124	0.126	0.147	**0.101**	0.125	0.126	0.126	0.154
Modified Adsorption	6.917	3.982	0.284	0.128	0.130	0.145	**0.105**	0.129	0.130	0.130	0.153

be close to their a-priori labels, (ii) nodes that are connected by larger weights should have high overlap in label distribution (iii) regularize estimated label distribution at every step.

3. EXPERIMENTS AND RESULTS

3.1. Intent Inventory Construction

To evaluate the quality of the learned inventory, we first cluster R_2 or R_2+R_3 using algorithms which do not require a predetermined number of clusters. Then, on 50 sequences randomly sampled from R_2 (with 18 manually created clusters as reference) we report normalized mutual information which is often used for evaluating soft-membership clusters [27].

We represent each activity in R_2 or R_3 with a vector composed of app vector $\vec{A}$ and description vector $\vec{D}$. The construction of these vectors is described in detail in Section 2.1.2. For apps $a_i \in A$, their vector representations $\vec{a_i}$ are computed as app-embedding via `app2vec`. For content word $w_i \in D$, we used pre-trained 300-dimensional Google News word2vec representation[2]. $\vec{a_i}$ are aggregated (and normalized) to form $\vec{A}$ and $\vec{D}$ is computed in the same way.

R_2 is clustered using affinity propagation (AP). We also augmented R_2 with conversational data R_3. The preference of each data point to be selected as exemplar in AP is varied, leading to different number of clusters. Evaluated on the aforementioned 50 samples, we find that augmenting R_2 with R_3 is better than using R_2 alone when the number of learned clusters is close to the ground truth (see Fig 4). Speech recognition hypotheses are not significantly inferior to manual transcripts. This indicates the effectiveness and robustness of incorporating atomic user utterances into learning complex user intents. Table 4 shows descriptions of examples clustered using the $R_2 + R_3$ model.

[2]`https://code.google.com/archive/p/word2vec/`

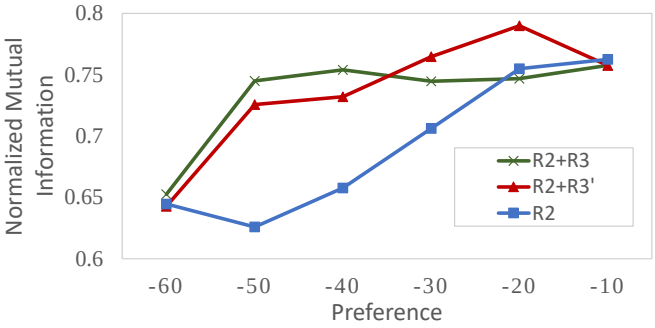

Fig. 4. Evaluation using different resources to learn R_2 clustering. 50 samples are randomly selected from R_2 and manually clustered. R_3' uses ASR hypotheses. When preference $\in [-50, -30]$, the number of clusters generated by Affinity Propagation is close to the ground truth (# clusters = 18).

3.2. Intent Inventory Update

Obtaining supervision is expensive. In real-life, when user performs a sequence of actions, we want the agent to 1) understand the user intent (later confirm with user if necessary) and 2) remove the noisy actions from the observation. This allows the agent to add a clean data point to its inventory. In this section, we first describe our approach to recognizing user intents with minimum supervision. We then discuss our attempt to separate content actions from noise.

3.2.1. Intent Recognition with Weak Supervision

We experimented with graph-based intent recognition methods using R_2. 70% of the total 1089 sequences are used as seeds. Our task is to predict the cluster for the remaining 30%. Seed labels were obtained by clustering the training part with Affinity Propagation as described in section 2.1. In obtaining seed labels, we used both app sequence and task descriptions (with additional help from conversational data). Note, however, that in this recognition task we do not have access to descriptions — by observing user performing a series

Table 4. Clustering examples on R_2.

Cluster	Example Descriptions
1	playing crossy road with optimum ram; playing spiderman;
2	updating apps on Google play; updating apps
3	used voice app to navigate to hambones restaurant in Lawrenceville; using maps to look up a work-related phone number for a business and then call
4	trying to find an address for our campsite at cherry springs; using Google search and maps to look up estimated driving time from Pittsburgh to Cleveland connor theater.
5	choosing a picture to send on Whatsapp; sending pictures to friend

Table 5. Evaluation of BIO tagging. Overall precision, recall and F_1 are reported.

Feature	P	R	F
Majority	0.69	0.69	0.69
Category	0.73	0.69	0.70
AppID	0.74	0.70	0.71
app2vec	0.78	0.73	0.75
desc2vec	**0.78**	**0.75**	**0.76**

of actions, we want the agent to interpret the intent.

During graph construction, we used K=5 for all Nearest Neighbor methods, $\gamma = 0.5$ for all Kernel Methods and default values for all covariance estimation techniques as set in `scikit-learn` [28]. For graph-labeling, we ran the random-walk for 1000 iterations in all three settings.

For each testing app sequence, our model estimates a distribution over labels. To evaluate the performance of SSL methods, we compute Kullback-Leibler (KL)-divergence between this distribution and the cluster distribution obtained in section 2.1. As we can see in Table 3, the `rbf` kernel-based graph construction with label propagation approach outperforms others. We note that graphs constructed through kernel-based methods and covariance methods have higher edges compared to nearest-neighbor based methods. Also, kernel based methods generate graphs with fewer nodes with high degreeness i.e., few nodes with higher influence. Whereas, graphs generated by covariance methods have evenly distributed node degreeness. We believe that kernel-based methods perform better because of higher degreeness thus propagating high quality and low-noisy labels to the unlabeled nodes. Based on the recognition confidence, agent can perform the actions in step 2 in Fig 1 (e.g., confirming the predicted intent). Later, step 3 can be carried out.

3.2.2. Content Apps Extraction

We adopted sequence labeling on any observed app sequence. The task is to tag each app with one of B (begin), I (in) and O (out) labels. By accurately identifying B's and I's, the agent demonstrates the ability to extract relevant content apps. We trained our Conditional Random Fields-based BIO tagger on R_1. Features for each app can be its category (e.g., games or communication) or app identity. Alternatively, we also used vector representations of app identity (the aforementioned `app2vec`) and app descriptions provided by the app store. We used `doc2vec` in `gensim` [29] to learn the description vector. As shown in Table 5, using a vector representation of apps can achieve better overall performance. This shows the feasibility of separating content apps from background or noisy apps in observed user activity.

4. CONCLUSION

Our long-term goal is to develop an agent that learns how to perform a complex task by decomposing it into smaller atomic tasks and execute in sequence. In this paper, we describe a framework that enables an agent to infer a user's intent while they use applications on their smart phone. To this end, our agent can learn an inventory of intents from a small set of task-oriented user utterances. We show that on previously unseen user activities, the agent reliably recognizes user intents using graph-based semi-supervised learning methods. We also demonstrate that our CRF-based sequence labeling model can effectively segment a sub-sequence of applications, that potentially map to a user intent, from a larger sequence of applications. Finally we made our dataset, models and the system output available to the research community.

5. ACKNOWLEDGMENT

This work was supported in part by the Yahoo InMind project at Carnegie Mellon and by the General Motors Advanced Technical Center–Israel.

6. REFERENCES

[1] Ming Sun, Yun-Nung Chen, and Alexander I. Rudnicky, "An intelligent assistant for high-level task understanding," in *IUI*, 2016.

[2] Toby Jia-Jun Li and Brad Myers, "Smartphone text entry in cross-application tasks," in *CHI Workshop on Inviscid Text Entry and Beyond*, 2016.

[3] Yun-Nung Chen, Ming Sun, and Alexander I. Rudnicky, "Leveraging behavioral patterns of mobile applications for personalized spoken language understanding," in *ICMI*, 2015, pp. 83–86.

[4] Aasish Pappu and Alexander Rudnicky, "Predicting tasks in goal-oriented spoken dialog systems using semantic knowledge bases," in *SIGDIAL*, 2013, pp. 242–250.

[5] Aasish Pappu and Alexander I Rudnicky, "Learning situated knowledge bases through dialog," in *Interspeech*, 2014, pp. 120–124.

[6] Ming Sun, Yun-Nung Chen, Zhenhao Hua, Yulian Tamres-Rudnicky, Arnab Dash, and Alexander I. Rudnicky, "Appdialogue: Multi-app dialogues for intelligent assistants," in *LREC*, 2016.

[7] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in NIPS*, 2013, pp. 3111–3119.

[8] Ma Qiang, S. Muthukrishnan, and Wil Simpson, "App2vec: Vector modeling of mobile apps and applications," in *ASONAM*, 2016.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan, "Latent dirichlet allocation," *JMLR*, vol. 3, no. Jan, pp. 993–1022, 2003.

[10] Fabio Gagliardi Cozman, Ira Cohen, Marcelo Cesar Cirelo, et al., "Semi-supervised learning of mixture models," in *ICML*, 2003, pp. 99–106.

[11] Thorsten Joachims, "Transductive inference for text classification using support vector machines," in *ICML*, 1999, vol. 99, pp. 200–209.

[12] Xiaojin Zhu and Andrew B Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[13] Yves Grandvalet and Yoshua Bengio, "Semi-supervised learning by entropy minimization," in *NIPS*, 2004, pp. 529–536.

[14] Avrim Blum and Shuchi Chawla, "Learning from labeled and unlabeled data using graph mincuts," in *ICML*, 2001.

[15] Xiaojin Zhu, Zoubin Ghahramani, John Lafferty, et al., "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003, pp. 912–919.

[16] Gad Getz, Noam Shental, and Eytan Domany, "Semi-supervised learning–a statistical physics approach," *arXiv preprint cs/0604011*, 2006.

[17] Thorsten Joachims et al., "Transductive learning via spectral graph partitioning," in *ICML*, 2003, pp. 290–297.

[18] Xianpei Han, Le Sun, and Jun Zhao, "Collective entity linking in web text: a graph-based method," in *34th ACM SIGIR*, 2011, pp. 765–774.

[19] Delip Rao and Deepak Ravichandran, "Semi-supervised polarity lexicon induction," in *EACL*, 2009, pp. 675–682.

[20] Bhavana Dalvi, Einat Minkov, Partha P Talukdar, and William W Cohen, "Automatic gloss finding for a knowledge base using ontological constraints," in *WSDM*. ACM, 2015, pp. 369–378.

[21] Tony Jebara, Jun Wang, and Shih-Fu Chang, "Graph construction and b-matching for semi-supervised learning," in *ICML*. ACM, 2009, pp. 441–448.

[22] Jon Louis Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[23] Olivier Ledoit and Michael Wolf, "A well-conditioned estimator for large-dimensional covariance matrices," *Journal of multivariate analysis*, vol. 88, no. 2, pp. 365–411, 2004.

[24] Xiaojin Zhu and Zoubin Ghahramani, "Learning from labeled and unlabeled data with label propagation," Tech. Rep., CMU CALD-02-107.

[25] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly, "Video suggestion and discovery for youtube: taking random walks through the view graph," in *Proceedings of the 17th WWW*. ACM, 2008, pp. 895–904.

[26] Partha Pratim Talukdar and Koby Crammer, "New regularized algorithms for transductive learning," in *Joint ECML/KDD*. Springer, 2009, pp. 442–457.

[27] Aaron F McDaid, Derek Greene, and Neil Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv preprint arXiv:1110.2515*, 2011.

[28] 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," *JMLR*, vol. 12, pp. 2825–2830, 2011.

[29] Radim Řehůřek and Petr Sojka, "Software Framework for Topic Modelling with Large Corpora," in *LREC*, 2010, pp. 45–50.