Ronnakorn Rattanakornphan

CSCI6961: Deep Learning

Final Project Report

Note: project code is located at https://github.com/ronnakorn9/AutoLabel/
Please see main.ipynb for detailed process

1. Introduction

Allowing a machine to complete a task via human language is an important and valuable application of Artificial Intelligence (AI). A user could ask a phone, using voice, to display a map route to his workplace. A developer could ask a chatbot to help generate code structure for their program. A customer could ask a booking application to reserve a flight on Monday. These examples are only a small fraction of how we use human language to interact with machines and obtain a desired result (ex. getting a flight ticket). However, the process the AI must go through to interpret human language efficiently, the "learning", requires a significant investment.

To achieve training data that allow AI to learn and understand human language, a sufficiently large dataset is essential. The process usually involves humans manually categorizing human language data with the intended label. This method is time-consuming, especially considering that the label would need to be remade for each dataset.

Particularly, we are interested in datasets for user intent detection, a task of determining user purpose or goal, given user query or context. This task has very useful applications in search and user recommendations.

To alleviate the issue, we proposed a method that automatically labels user intent in text datasets, using a combination of clustering method and Natural Language Understanding (via transformer models). Our method is an unsupervised approach for automatically detecting and labeling user intent.

2. Related Work

Chatterjee & Sengupta (2020) achieve intent mining by

- Using a universal-sentence-encoder to embed the text into a numeric vector

- Using a proposed clustering method, ITERDBSCAN, to do clustering on embedded vector
- The paper then proposed to label some of the representatives on each cluster
- Finally, propagate the label of cluster representative to the rest of the members in dataset
    - The propagation process could be simply applying the same cluster to all members of the model or training a classification model based on a set of known intent dataset

3. Method

The process of our proposed method can be divided into 3 subprocess

1. Embedding
2. Clustering
3. Generating label candidate
    a. Remove entity
4. Zero-shot inference

3.1 Embedding

The first process of our auto-clustering method is to do an embedding on the text data. The goal is to transform the text data into numeric vectors so that we can run clustering algorithm on them.

For sentence embedding, we use Sentence-BERT (SBERT)[1] which is a modified version of BERT transformer model with siamese and triplet network structures. Particularly, we use all-MiniLM-L6-v2 model from Python's SentenceTransformers framework. The model is pre-trained on more than 1 billion English sentence pairs. We do not run any further fine-tuning on the embedding model. The output of the embedding model is a dense 384-dimension vector.

[1] model can be found at https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

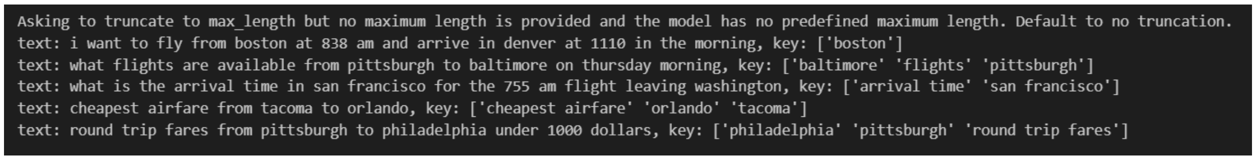
3.2 Clustering

The second process is to run the clustering algorithm. The goal is to group the text with similar semantics together. While most clustering algorithm could achieve this, we opt for DBSCAN (with cosine distance metric) since it can handle unbalanced clusters, a common situation for text datasets.

3.3 Generating label candidate

In the third process, we generate a list of candidate intent for the text corpus. We accomplish this by first sampling text data for each of the cluster. Then, using a transformer model, we extract keywords from each of the sampled text data. Finally, we merge all the obtained keywords by computing a set of smallest substrings from all the keywords, for example, a list of [“transport”, “ground transport ”, “ground transportation”] is reduced to just [“transport”].

We use KBIR-OpenKP[2] for keyword extraction. It is a Keyphrase Boundary Infilling with Replacement (KBIR) model built on RoBERTa architecture and modified by “adding an Infilling head and a Replacement Classification head that is used during pre-training.” The model is pre-trained on OpenKP dataset for token classification task.



```
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.
text: i want to fly from boston at 838 am and arrive in denver at 1110 in the morning, key: ['boston']
text: what flights are available from pittsburgh to baltimore on thursday morning, key: ['baltimore' 'flights' 'pittsburgh']
text: what is the arrival time in san francisco for the 755 am flight leaving washington, key: ['arrival time' 'san francisco']
text: cheapest airfare from tacoma to orlando, key: ['cheapest airfare' 'orlando' 'tacoma']
text: round trip fares from pittsburgh to philadelphia under 1000 dollars, key: ['philadelphia' 'pittsburgh' 'round trip fares']
```

Fig.1. Example of keyword extraction

To remove any entity name, ex. name of an airport, we use another transformer model to mark the word token with entity name for removal. We use DistilBERT base model (uncased)[3] fine-tuned for Named Entity Recognition task with CoNLL-2003 dataset.

After this process, we obtain a list of text label candidates generated from text samples derived from each text cluster.

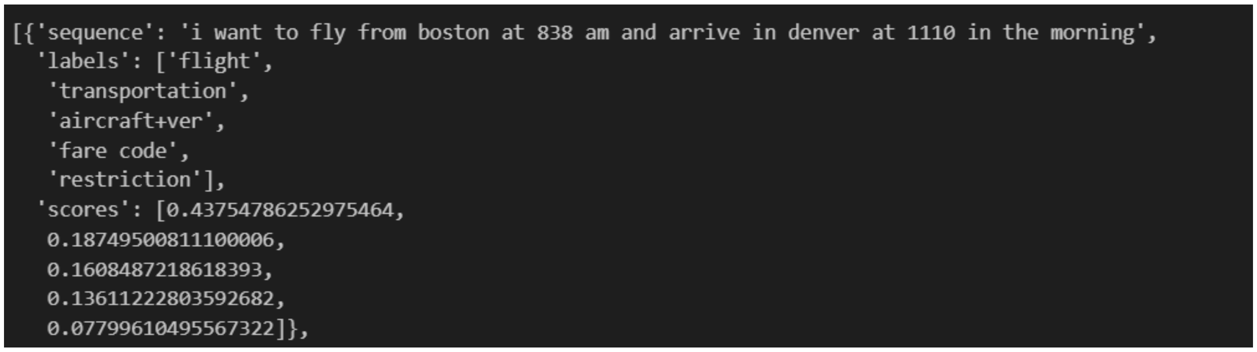
[2] model can be found at https://huggingface.co/ml6team/keyphrase-extraction-kbir-openkp
[3] model can be found at https://huggingface.co/dslim/bert-base-NER-uncased

3.4 Zero-shot inference

Finally, to propagate the text label (the candidate intent) to the cluster, we run class inference using a zero-shot classification model. In other words, we run a classification task with a zero-shot model to pair each sentence with one of the candidate labels.

We use BART (large) model trained on Multi-Genre Natural Language Inference (MultiNLI)[4] dataset as our zero-shot text classification model.

After this process this one, we paired each text sentence with the corresponding candidate text label. See Fig. 6 for example result



```
[{'sequence': 'i want to fly from boston at 838 am and arrive in denver at 1110 in the morning',
  'labels': ['flight',
   'transportation',
   'aircraft+ver',
   'fare code',
   'restriction'],
  'scores': [0.43754786252975464,
   0.18749500811100006,
   0.1608487218618393,
   0.13611222803592682,
   0.07799610495567322]},
```

Fig.2. Example output of zero-shot classification. The input to the classifier is a text sentence and a list of possible labels. The model will assign each label probability for that sentence. In this example, “i want to fly from boston at 838 am and arrive in denver at 1110 in the morning” has a probability of 0.438 to be paired with the label “flight.” In our work, we assign the label with the highest probability to each sentence.

[4] model can be found at https://huggingface.co/facebook/bart-large-mnli

Experiment

We run our auto-labeling method on the ATIS dataset[5], comparing it against DBSCAN and ITERDBSCAN.

The metrics we used for performance measurement are

- Homogeneity score
- Completeness score
- Normalized mutual information (NMI)
- Adjusted mutual information
- Adjusted rand score

The clustering method was run on the embedding vector from 3.1.

For DBSCAN, we use the method in Python sklearn's library with eps = 0.2 (max neighbor distance), min_samples = 10, and cosine distance metric.

For ITERDBSCAN, we use the implementation of Chatterjee and Sengupta (2020). We run 100 iterations of hyperparameter random search and select the one with the best NMI score. The final hyperparameters are initial_distance=0.1126, initial_minimum_samples=10, delta_distance=0.015, delta_minimum_samples=2, max_iteration=40.

For our Auto-labeling, we use the initial cluster result from the aforementioned DBSCAN.

[5] Dataset can be found at https://github.com/howl-anderson/ATIS_dataset/blob/master/README.en-US.md

Result

We see a noticeable improvement over the original DBSCAN result, both in clustering metrics and similarity visual to the true cluster. See Fig. 5 for details. Compared to ITERDBSCAN, we can achieve better clustering performance while maintaining a reasonable amount of cluster.

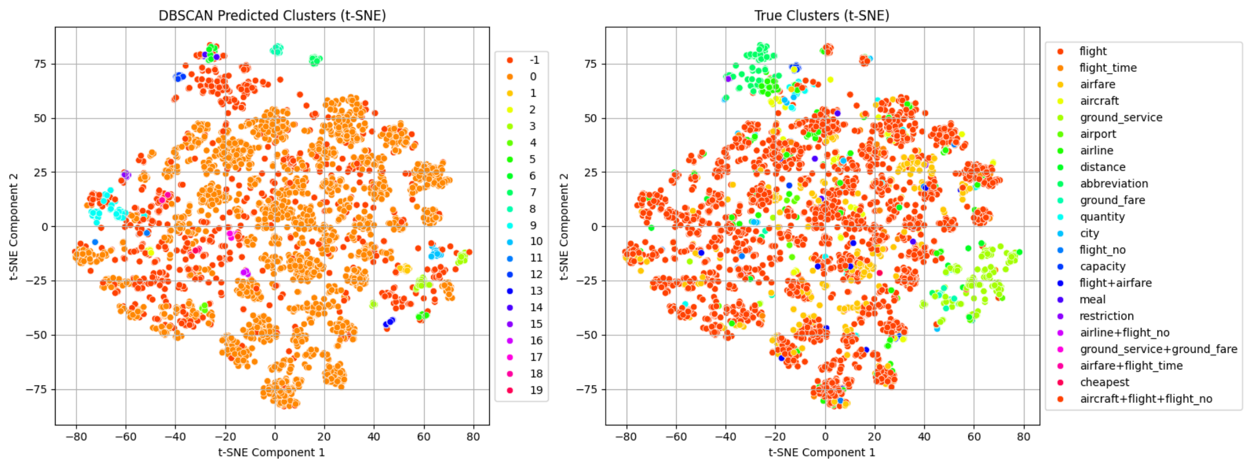


Fig.3 DBSCAN clustering result compared to the true cluster.

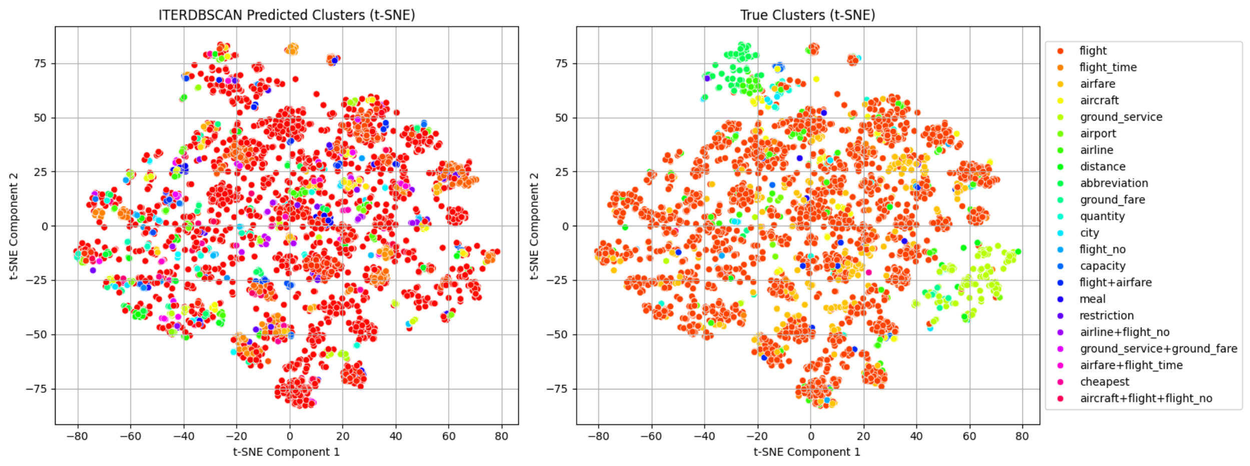


Fig.4 ITERDBSCAN clustering result compared to the true cluster. The label of ITERDBSCAN is omitted because it contains too many cluster id (about 300 of them).

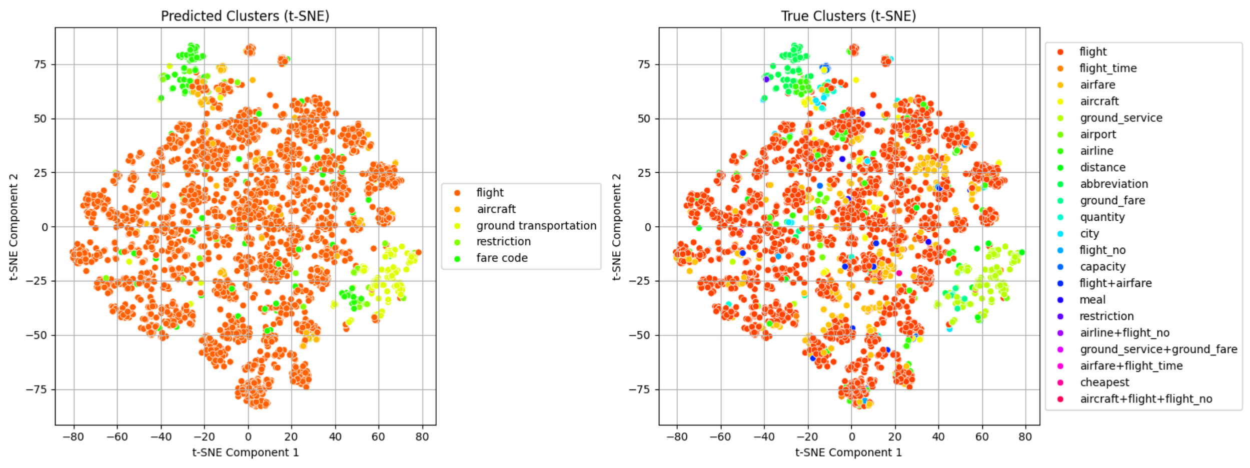
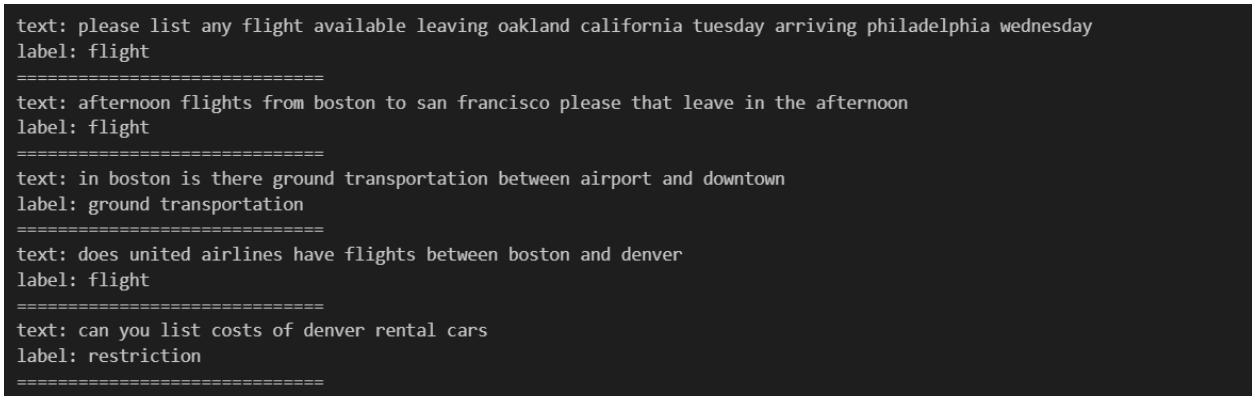


Fig.5 Auto-Label clustering result compared to the true cluster. The label are more

score\method	DBSCAN	ITERDBSCAN	Auto-Label
Homogeneity score	0.14	0.34	**0.33**
Completeness score	0.29	0.11	**0.65**
Normalized mutual information (NMI)	0.19	0.17	**0.44**
Adjusted mutual information	0.17	0.11	**0.44**
Adjusted rand score	0.13	-0.06	**0.49**

Table 1: Clustering metrics comparison between DBSCAN, ITERDBSCAN, and Auto-label (this work)



```
text: please list any flight available leaving oakland california tuesday arriving philadelphia wednesday
label: flight
==============================
text: afternoon flights from boston to san francisco please that leave in the afternoon
label: flight
==============================
text: in boston is there ground transportation between airport and downtown
label: ground transportation
==============================
text: does united airlines have flights between boston and denver
label: flight
==============================
text: can you list costs of denver rental cars
label: restriction
==============================
```

Fig.6. An example output of Auto-labeling. While it can accurately cluster and label most of the dataset, it still struggles to label a smaller intent set (ex. label = restriction).

Conclusion

In this work, we present Auto-label, an unsupervised approach to automatically label unseen data. The biggest advantage of this work is to obtain a labeled dataset from an unlabeled one without human intervention. The method uses a combination of clustering algorithm and transformer models. In our experiment comparison with other clustering methods, we achieve significantly higher performance metrics.

While Auto-label can achieve automatic labeling on unseen data, it is not without disadvantages.

First, this method requires a total of 4 different transformer models, making it quite slow to compute and requiring much more memory for those additional models.

Second, the usage of keyword extraction means that the text label must be a substring of existing text. For example, a user query of “Which road do I take to evade traffic?” would be labeled as “road” or “traffic” rather than “navigation.”

For our further work, we hope to improve the keyword extraction so that it can infer a meaningful keyword rather than being restricted to a substring of existing text. Another improvement we would like to try is to mask the entity in the original text with a common term instead. For example, “When is the flight from Boston to California?” would become “When is the flight from [location1] to [location2]?”

Reference

Chatterjee, A., & Sengupta, S. (2020). Intent mining from past conversations for conversational agent. *arXiv preprint arXiv:2005.11014*.

Hulth, A. (2003). Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing* (pp. 216-223).

Kulkarni, M., Mahata, D., Arora, R., & Bhowmik, R. (2021). Learning rich representation of keyphrases from text. *arXiv preprint arXiv:2112.08547*.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., ... & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Reimers, N., & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.