

Identify Semantically Similar Queries in Virtual Assistant Dialogues

Stanford CS224N Custom Project

Weiwei Zhang

Department of Computer Science
Stanford University
wwzhang@stanford.edu

Abstract

Currently for production virtual assistants, it's hard to accurately understand feature usages from usage logs, because if a query is not parsed with the correct natural language (NL) intent in the existing system, then it's unlikely to be counted towards the correct feature's usage downstream. We wanted to build on top of Sentence-BERT (SBERT) [1], a finetuned Bert model proven to have great performances in common semantic textual similarity (STS) tasks, and use it to find similar user queries given an example feature query in the raw text usage logs directly. We curated a custom dataset from Google's Taskmaster¹ dataset as our virtual assistant dialogue corpus. We ran a baseline model which is an SBERT variant off-the-shelf on this new dataset, and we tried finetuning SBERT variants on our custom dataset to get better performances. In addition, we experimented various ways to leverage entity related information in the queries to improve semantic similarity search quality, and we've found a named entity recognition (NER) technique called E-BERT-concat (Poerner, 2020)[2] to visibly improve model performances.

1 Key Information to include

- Mentor: Megan Leszczynski
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

Since SBERT has demonstrated good results on semantic textual similarity (STS) benchmark tests, we wanted to see if it can help us find user utterances of the same feature in a dialogue dataset, given an example query of that feature. All SBERT model variants were trained and evaluated on formal written texts, such as Stanford Natural Language Inference (SNLI) Corpus², various STS benchmark data sets, Microsoft Research Paraphrase Corpus[3], etc. However, we'd like to argue that virtual assistant dialogues are distinct in their own ways and therefore worth experimenting this model on. They are more colloquial and may contain abbreviations, filler words, etc. They may even contain grammatical or spelling errors. SBERT generates meaningful sentence embeddings that can be directly compared for similarity using cosine similarity. This offers a computationally efficient way to analyze raw user queries directly to find usage patterns. This can in turn help with feature metrics collection and new feature discovery which then can help improve overall virtual assistant usability.

¹<https://github.com/google-research-datasets/Taskmaster>

²<https://nlp.stanford.edu/projects/snli/>

Sentence 1	Sentence 2	Similar?
The euro rose above US\$1.18, the highest price since its January 1999 launch.	The euro rose above \$1.18 the highest level since its launch in January 1999.	Yes

Table 1: Example from Microsoft Research Paraphrase Corpus

Sentence	Feature Label
Hey I need to take my truck in <u>ASAP</u> . Could you get me an appointment at Intelligent Auto?	auto-repair-appt
Hey Donna, I'm driving <u>right now</u> can you make me a car appointment?	auto-repair-appt

Table 2: Example from our custom dialogue dataset. Note that we also manually labeled each query with a feature label, indicating what task the user asked the assistant to achieve.

We ran one baseline model directly provided by SBERT authors³. Then we tried finetuning a variety of pretrained SBERTs on our dataset for different experiment goals. Because of how different our dialogue dataset is comparing to the datasets pretrained SBERT models trained on, we thought that finetuning on our custom dataset should greatly improve model performances. After finetuning, we did some early error analysis and posed two questions for further improvements:

1. does having world knowledge of entity meanings help semantic similarity search?
E.g. if you know "noodles" means a type of food, then someone asking for "noodles" is more likely trying to order food than to buy movie tickets.
2. given that the entities in a sentence are often strong indicators of what the sentence means (e.g. if a user wants to book a hotel, the sentence will likely have the word "hotel" or a specific hotel brand; if a user wants to buy movie tickets, the query will likely contain an entity that's a movie name or movie star name), does emphasizing the entity substring in an input text string somehow help improve semantic search accuracy?

For the first question, we ran comparison between two pre-trained SBERT models finetuned on our dialogue dataset, one pretrained on paraphrase datasets for paraphrase mining, and another pretrained on MS MARCO dataset⁴ for question answering (QA). Then we ran a knowledge testing experiment on the QA model to see if this model saved entity related knowledge during pretraining that could improve semantic similarity search performance. For the second question, we tried adapting the E-BERT-concat NER technique (Poerner, 2020)[2] in a pretrained SBERT model for paraphrase mining. We found that this NER technique did indeed improve the semantic similarity search results. Our code is on github: <https://github.com/weiwzhang/utter-search>.

3 Related Work

3.1 Generate meaningful sentence embeddings

A key step to semantic similarity search is creating meaningful sentence embeddings. Our project mainly bases off SBERT[1], a finetuned BERT model using siamese and triplet network structures in order to derive semantically meaningful embeddings that can be compared using cosine-similarity. We used the siamese SBERTs in our project as baselines. Siamese SBERT training inputs are pairs of paraphrases, and each sentence in a pair is passed through a BERT network with same parameters to compute embeddings. Then a cosine similarity loss is computed between them and back-propagate through the network. It can find semantically similar sentence pairs at inference using significantly less computation, because in original BERT model you need to pass in every pair of sentences directly to find out their similarity scores, which is in $O((n)^2)$ time; but for SBERT we just pass each individual sentence into the model once, and then we can compare output embeddings directly using their cosine similarity scores, so it's in $O(n)$ time. There are also some other popular sentence embedding models, such as Google's Universal Sentence Encoder (USE) [4] which trains a

³https://www.sbert.net/docs/pretrained_models.html

⁴<https://github.com/microsoft/MSMARCO-Passage-Ranking>

transformer network and augments unsupervised learning with training on SNLI, and Facebook’s InferSent[5] which trains a siamese BiLSTM network with max-pooling over the output. However, SBERT outperforms them significantly in various STS tasks[1], which was why we chose it as the basis of our project. Our project added on to the research by transferring SBERT onto a new, more colloquial virtual assistant dialogue dataset, as well as exploring combining NER techniques with existing SBERT models.

3.2 Named Entity Recognition Techniques

Part of our project explores using NER techniques to improve our semantic similarity search task. Specifically, we were looking for previous work that can parse input texts and tag entities well. We researched some common libraries, such as spaCy⁵ and flair[6]. We prioritized ease of use and availability of a relatively larger set of entity tags, therefore we ended up using spaCy package over flair. Note that to limit the scope of our project, we considered but didn’t explore whether the granularity to which we tagged entities could impact model performances or not. For example, Ling, 2012[7] proposed FIGER with 112 entity tags, and this tagging system has shown improvement in tasks such as building relation extraction system. We didn’t explore using FIGER or other fine grained tagging since there was no readily available fine-grained tagging packages we could use.

4 Approach

Given an input query in the virtual assistant dialogue dataset, we wanted to find other semantically similar queries that were asking for the same feature, e.g. given "find starbucks", can we find other queries such as "find coffee shops" or "find Peet’s coffee". Our definition of "semantically similar" is special in that we care more about whether two queries are trying to accomplish the same task rather than having exactly the same meaning. E.g. "find starbucks" and "find coffee shops" would be considered semantically similar in our case, even though one is asking for only Starbucks while the other means to find coffee in general. We curated our dataset so that each utterance would have a feature label, and having the same feature label is equivalent to being semantically similar.

Our baseline model is an SBERT variant paraphrase-distilroberta-base-v1⁶ pretrained specifically for paraphrase tasks, which we used as-is. All other models are SBERT variants finetuned on our dataset and evaluated for performance on our hold-out test set. At training, we fed pairs of sentences and their similarity scores into SBERT. We used a siamese SBERT structure, passing each sentence in a pair through the same BERT network and generating respective embeddings. Then we computed a cosine similarity score between two embeddings in each pair, and calculated mean squared error loss (MSE) between predicted score and true similarity scores: $\min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_{sim} - \cos(\text{similarity}(u, v)))^2$ (u, v are embeddings, n = total data size). Our training objective is to minimize MSE loss.

At inference, we used our BERT model to first generate all test corpus embeddings. Then we used each embedding to find top K most similar queries in the corpus to the input query (we’ll explain choice of K later). After that, we evaluated the quality of similar queries found against the gold labels using F1 metrics.

Note that data cleaning, model finetuning, and testing and evaluation are all original code. We used SentenceTransformers APIs⁷ to build the finetuning pipeline, which simplified back-propagation and input text tokenization.

5 Experiments

5.1 Data

We used Google’s Taskmaster-1 and Taskmaster-2 datasets⁸, consisting of 25k dialogues. Each dialogue contains a series of interactions between a user and a virtual assistant, where the user is trying to achieve some task. In each dialogue, each utterance is tagged with speaker (user or assistant),

⁵<https://spacy.io/>

⁶https://www.sbert.net/docs/pretrained_models.html

⁷<https://www.sbert.net/docs/quickstart.html>

⁸<https://github.com/google-research-datasets/Taskmaster>

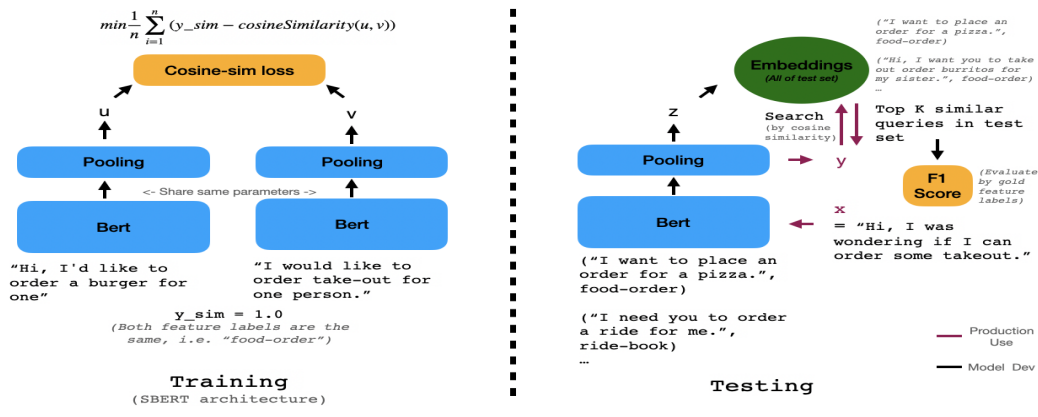


Figure 1: Model Overview

instruction id⁹ (task to achieve), and other annotations. We processed this corpus into our desired format with the following schema: utterance (string, user’s query), and feature label (string, user’s task to achieve). We manually curated our final dataset by pulling one user query out of each dialogue that best represents user’s intended feature to use. We also created unified feature labels for all queries, and performed other data cleaning such as de-duping, removing null values, correcting wrongly annotated data points, etc.

Overall we used two versions of datasets. We first used Data-full with more labels. After noticing the issues of data overfitting and ambiguous feature labels in early trainings, we trimmed our data and re-split the train and test datasets, to create our final data version Data-final.

- **Data-full** a total of 17505 sentences and 11 feature labels. Out of these sentences, we randomly sampled and created a training corpus of around 580k pairs of paraphrases(70% are unique pairs of similar utterances sharing same feature label, 30% pairs share different labels). In remaining data, we randomly sampled a 20k-pair dev data(similar vs. dissimilar is roughly 7:3 as well), and a test dataset of 5k utterances. All 11 features are present in both training and testing datasets, and each of the 11 feature labels has the same amount of data present in test corpus (470 data points per label). For training and dev datasets, we assigned a similarity score of 1.0 for each pair of sentences that shared same feature label, and 0.2 for each pair with different feature labels. The motivation was to push the difference between similar vs. dissimilar sentences relatively large.
- **Data-final** 550k pairs in training corpus (similar vs. dissimilar is again roughly 7:3) containing 6 labels ('auto-repair-appt', 'coffee-order', 'flight-search', 'food-order', 'movie-search', 'ride-book'), 26k dev corpus (similar vs. dissimilar is roughly 8:2), and 3760 sentences in test corpus. Both dev and test corpuses have 8 labels (all labels in training + 'hotel-search', 'music-search'). Similarly, in the test corpus each label has exactly 470 data points. Note that we have fewer feature labels in Data-final comparing to Data-full, because we wanted to remove some more ambiguous labels (e.g. restaurant-book, food-order and restaurant-search could often have overlapping utterances, which complicated our experiments, so we only kept food-order).

5.2 Evaluation method

We used F1 score to measure model performance. During testing, for each sentence we computed precision (out of top K similar queries, how many shared the same feature label as input query) and recall (out of all queries in the corpus that shared the same feature label as input query, how many were selected in top K), and then computed $f1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$. Note that to simplify our analysis,

⁹Note that each version of Taskmaster dataset has slightly different schema, but they all have variants of instruction tags as described above.

we chose $K = 470$ for all experiments using Data-full and Data-final, because we had 470 data points per feature label in test set.

We also did qualitative analysis by pulling random samples of selected utterances and grading them manually.

5.3 List of Models

5.3.1 Baselines

We had one **Baseline** model: a pre-trained SBERT variant paraphrase-distilroberta-base-v1¹⁰, without any fine-tuning on our custom dataset. It is built on top of Huggingface's RoBERTa model and finetuned on large scale paraphrase dataset. We chose this variant because it's one of the best-performing models in terms of both quality and speed on STS and paraphrase tasks. It's trained with a batch-size of 16, hidden_size=768, num_hidden_layers=6, num_attention_heads=12, hidden layer dropout_prob=0.1 (both hidden and attention).

5.3.2 Fine-tuned Models

Here's a full list of finetuned model configurations for different experiment purposes. All of them took roughly 24 hours to finetune¹¹.

- **Paraphrase-Finetune** same SBERT variant paraphrase-distilroberta-base-v1 finetuned on our custom dataset. We finetuned with batch size = 32, epochs = 4, and learning rate = 1e-3.
- **QA-Finetune** another SBERT variant msmarco-distilroberta-base-v2 finetuned on our custom dataset. The original SBERT model was pretrained for information retrieval tasks on 500k data points from MS MARCO¹² dataset (actual Bing search queries). We finetuned this model on our custom dataset using a siamese structure and cosine similarity loss, same for paraphrasing tasks. Pretraining configurations were the same as Baseline, and finetuning configurations same as Paraphrase-Finetune.
- **Paraphrase-Corrupted** same model set up as Paraphrase-Finetune, but finetuned on a modified version of our dataset, where we corrupted the data by masking out entities to test whether pretrained models retained world knowledge of entities that would help in our semantic similarity search task. More details in next section.
- **QA-Corrupted** same model set up as QA-Finetune, but finetuned on the same corrupted data that Paraphrase-Corrupted was trained on.
- **Paraphrase-NER** same model set up and dataset as Paraphrase-Finetune, but we incorporate E-BERT-concat technique (Poerner, 2020)[2] to preprocess the dataset as an attempt to improve model performance.

5.4 Experiment Results and Analysis

Experiment 1: Baseline vs. Finetune on custom dataset First we wanted to engage how good the baseline model could perform on our dataset with and without finetuning on our dataset (figure 2). Comparing Baseline and Paraphrase-Finetune, we can see that finetuning greatly improved model performance. However, the fact that Paraphrase-Finetune achieved almost perfect F1 score on most feature labels suggested that we may be overfitting. It may be due to how Taskmaster dataset is created. For each dialogue, the two speakers (one actual user and another person imitating a voice assistant) are given very specific instructions, in some cases as detailed as which area to search food for or which auto repair shop to book appointment at. Therefore there's limited variation between train and test datasets for some feature labels. This finding prompted us to create Data-final, which has different labels present in training vs. testing sets.

From Baseline's F1 graph, we see low performance and spread-out F1 distribution in restaurant-search and restaurant-book. A likely cause was those two labels had overlapping utterances, e.g. when a

¹⁰https://www.sbert.net/docs/pretrained_models.html

¹¹on a NC6 promo VM on Microsoft Edge

¹²<https://microsoft.github.io/msmarco/>

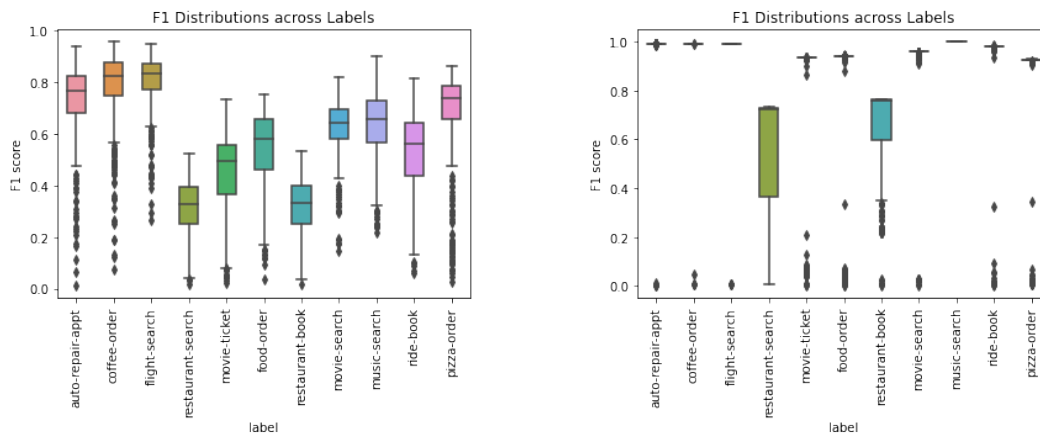


Figure 2: Baseline vs. Paraphrase-Finetune (left to right) model F1 (on Data-full)

user says *"find me some Mexican food"* or *"I'm hungry"*, they could be searching for a restaurant, or trying to book one. This suggested an ambiguity flaw in how we defined our feature label categories. To avoid such ambiguities in our analysis, we ended up removing restaurant-search, pizza-order and restaurant-book, just kept food-order; we also removed movie-ticket and kept movie-search in Data-final. Note that all following experiments were ran on Data-final only.

In qualitative analysis of Baseline, we saw that the model may merely be recognizing identical phrases to the input query in test dataset and not understanding key feature-related information such as the entity. For example, top similar utterances of *"Help me to book a nice restaurant."* include *"I would like to book a flight, please."*, which showed that the model didn't understand the difference between "restaurant" and "flight" semantically. Model also performed poorly in music-search category, possibly because model didn't focus on entity information in the sentences enough, e.g. to understand *"Play something by Kendrick Lamar."*, you need to know Kendrick Lamar is a musician. One possible solution is to incorporate named entity recognition techniques to improve our model in understanding and recognizing entities in the data. Thus we explored following two experiments.

Experiment 2: Using world knowledge from pretraining to improve performance One idea we explored was, if we used a model pretrained on web search data for question answering, which may theoretically pick up world knowledge of entities in the dataset, and finetuned for semantic similarity search tasks, would it somehow be able to leverage any pretraining knowledge and have better performances in understanding sentences? To validate this theory, we introduced another SBERT variant trained for QA, msmarco-distilroberta-base-v2. We finetuned this model as well as Paraphrase-Finetune model on Data-final (figure 3).

First of all, we saw that Paraphrase-Finetune did very well on all feature labels present in both training and testing datasets, but had worse performances on feature labels only present in test dataset (hotel-search and music-search). This validated what we discovered in experiment 1 about models overfitting to our dataset due to limited data variation. Therefore hotel-search and music-search were the important indicators of how well the model could generalize to new data.

Comparing Paraphrase-Finetune and QA-Finetune, the results seemed indecisive: QA-Finetune seemed to do a little worse in hotel-search feature label but slightly better in music-search label. We performed another knowledge testing experiment: we took Data-final and create a corrupted version by masking out entities identified by spaCy library with generic entity masks for both training and testing datasets. The rationale was, if the pretrained model remembered any knowledge on a particular entity, by masking that entity out in finetuning and testing, that information would be completely lost and cannot be used during testing, and models would perform much worse. We graphed the F1 score between new models Paraphrase-Corrupted vs. QA-Corrupted (figure 5). QA-Corrupted indeed did much worse than QA-Finetune in hotel-search and music-search, but we could see that Paraphrase-Corrupted did noticeably worse than Paraphrase-Finetune as well. It seemed that this corrupted dataset overall lowered model performances, therefore it was hard to justify how much

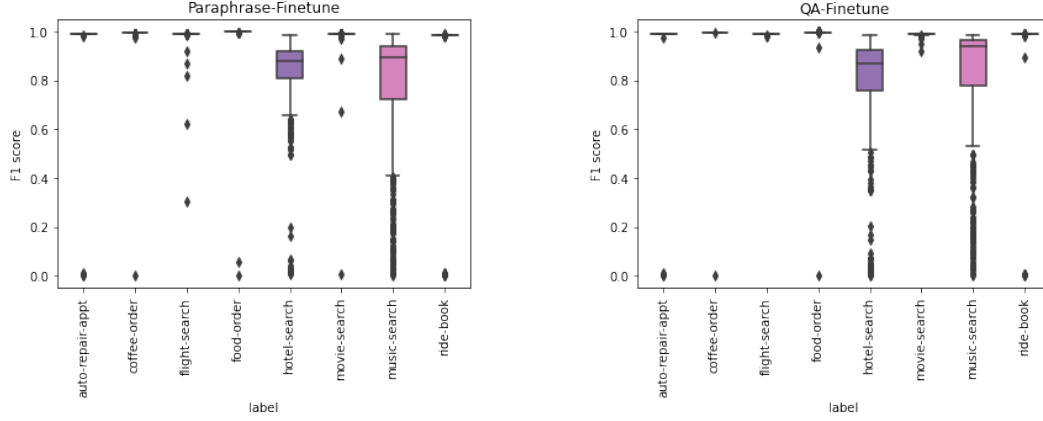


Figure 3: Paraphrase-Finetune vs. QA-Finetune F1 Score Distributions (on Data-final)



Figure 4: Experiment 2 - Knowledge Testing (on corrupted Data-final)

removing entities alone contributed to this degredation, and therefore hard to investigate whether pretrained QA model retained useful world knowledge on entities. A likely reason for overall model degredation is, after masking entities out, during testing there was less useful information in the sentences, therefore more difficult for model to make good predictions. Given all comparisons, we concluded that it was unclear whether pretrained model using web search data could retain useful world knowledge on entities that could help semantic similarity search.

Experiment 3: Using NER Tagging to improve performances We then explored whether tagging entities in input queries during training could improve model performance. Inspired by E-BERT-concat(Poerner, 2020)[2], we decided to preprocess training data by injecting entity tags from spaCy next to entities in input texts, as a way to help the model learn entity information. Note that unlike the original E-BERT-concat approach, we used coarse entity types from spaCy instead of Wikipedia entity IDs. We also didn't add special pretrained embeddings for entity tags in the model which was done in original E-BERT-concat method. Ideally if time permitted, we would also try to imitate these details in E-BERT-concat. We also considered replacing entities entirely with entity tags rather than concatenating them, but eventually decided to use concatenation because we wanted the model to

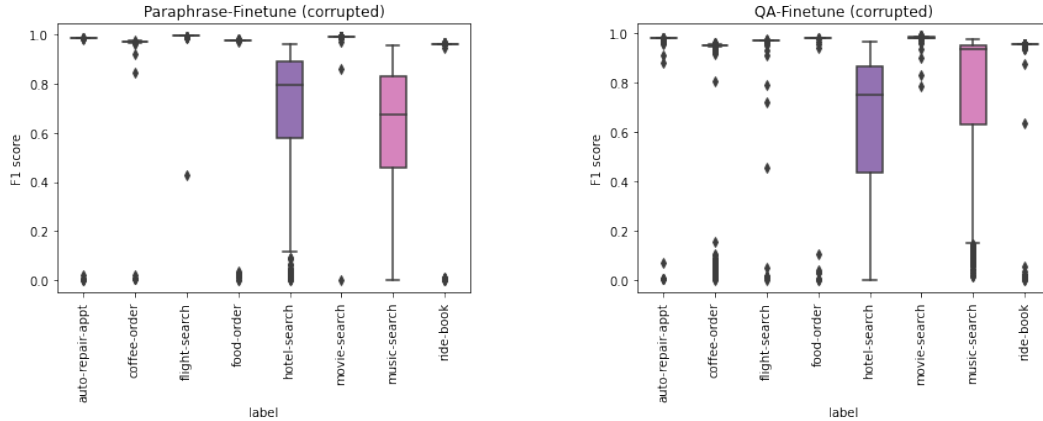


Figure 5: Paraphrase-Corrupted vs. QA-Corrupted F1 Score Distribution (Data-final)



Figure 6: Experiment 3 - E-concat during training (on Data-final)

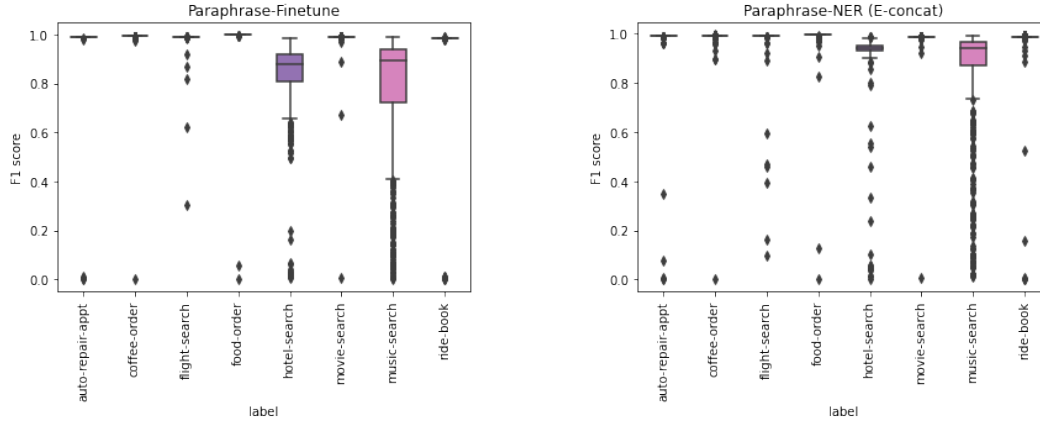


Figure 7: Paraphrase-Finetune vs. Paraphrase-NER F1 Score Distribution

have more useful information to learn during training. Also experiment 2 suggested that masking more information from training corpus didn’t help model performance. Figure 6 shows our E-concat process: we used spaCy to find entities in each input utterance in training corpus, and then concat each entity tag + slash (“/”) string in front of each entity. Similar to E-BERT-concat, we used slash because in BERT’s pretraining corpus, a slash is often used to separate two variants of the same word[8], so BERT should be able to understand that words before and after slash are semantically equivalent. Note that we didn’t do the same E-concat preprocessing for test corpus, because we wanted to prove that during finetuning our model was actually able to learn these entity information, and not just merely matching patterns like “<entity tag> /” during testing to help with performance.

We then finetuned Paraphrase-Finetune on the modified Data-final training set to create Paraphrase-NER model. Figure 7 showed Paraphrase-NER performance comparing to Paraphrase-Finetune. We saw that our E-concat technique visibly improved our model performance, suggesting that emphasizing entities during training helped model identify crucial information in testing sentences.

6 Conclusion

To recap, our goal was to find a good way to discover semantically similar feature queries in virtual assistant dialogues. We built on top of SBERT, and we tried multiple experiments to test how well SBERT transferred to our custom curated colloquial dialogue dataset. We also explored multiple ideas to improve model performance by leveraging entity related information in input texts. We’ve found entity tagging in input texts to be a great addition in improving model performances. A major limitation of our project is we could only curate limited virtual assistant dialogue data. It would be interesting to try our models on real life production data with more diverse feature labels and conversational patterns, and see if we can still get consistent results. In the future, we can do more analysis on why our QA based models didn’t perform better as we’d hypothesized. As a useful extension of this project, we can also explore using SBERT embeddings for natural language generation, since another real-life use case is to use sentence embeddings to generate more diverse test cases for yet-to-release new features, which wouldn’t have a lot of usage data to gather patterns and create enough test cases to robustly test features before public release.

References

- [1] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.
- [2] Nina Poerner, Ulli Waltinger, and Hinrich Schütze. E-BERT: Efficient-yet-effective entity embeddings for BERT. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 803–818, Online, November 2020. Association for Computational Linguistics.
- [3] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.
- [4] Daniel Cer, Yinfei Yang, Sheng yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal sentence encoder, 2018.
- [5] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data, 2018.
- [6] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.
- [7] X. Ling and D.S. Weld. Fine-grained entity recognition. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI)*, 2012.
- [8] Timo Schick and Hinrich Schütze. Bertram: Improved word embeddings have big impact on contextualized model performance, 2020.