Exploring the Effect of Semantic Similarity on Model Generalization

Stanford CS224N Custom Project

Hong Ju Jeon, Dustin Zubke
Department of Computer Science
Stanford University
hjeon@stanford.edu, dzubke@stanford.edu

Abstract

Research in instruction fine tuning (IFT) has shown significant potential to improve generalized model performance. To further the understanding of IFT, our work studies the relationship between semantic similarity of dataset tasks and model performance. We first develop an approach to calculating a measure of semantic similarity between tasks and then utilize the Sup-NatInst dataset[1] to construct training datasets with varying degrees of similarity. We find that performance improves both by decreasing the semantic similarity of the training tasks and by increasing the number of tasks. By using semantic similarity to group tasks, our semantically different TDiff-Instruct-200 model gets comparable performance to our semantically similar TSim-Instruct-300 model while using 1.5 times less data. We hope that this work can assist in building language models for low-resource tasks or languages by allowing researchers to use semantic similarity to identify new tasks with the greatest potential to boost model generalization.

Key Information: Custom Project Mentor is Siyan Li.

1 Introduction

Creating generalized models that can solve previously unseen tasks is an important problem in the field of Natural Language Processing (NLP). In recent years, language models have shown a remarkable ability to generalize across NLP tasks when using a pretrained, task-agnostic language model architecture on large amounts of data (Brown et al., 2022[2], Raffel et al., 2020[3]) To improve performance even further, work in instruction fine tuning (IFT) has demonstrated the potential to improve generalized model performance on unseen tasks (i.e. zero-shot) by leveraging the fact that NLP tasks can be described using natural language instruction (Wei et al., 2022[4], Ouyang et al., 2022[5]).

Having a large amount of data is often necessary to perform well at a task; however, not all tasks or languages have abundant data. In this paper, we make progress in the understanding of how IFT improves the model's generalization capabilities in low-resource environments by leveraging "semantic similarity" as measured through the cosine distance of embeddings. Specifically, we study the relationship between the semantic similarity of tasks in a group and the zero-shot performance of models finetuned on task groupings of varied semantic similarity.

Through our research, we find that finetuning models on training datasets with lower semantic similarity improve zero-shot performance. Furthermore, we add onto the insights gained by Wei et. al 2022[4] and Chung et al. 2022[6] to show that **decreasing semantic similarity is an alternative to increasing the number of task types or model size to increase performance**, if the dataset contains tasks that are highly similar to one another. We also find that the model improves in performance generally as we add more task types, but the rate of improvement is a function of the semantic similarities of those new tasks. Taking this approach further, we show that this trend holds for not only task types but also groupings based on semantic similarities of task categories.

Using our findings, we train a TDiff-Instruct-200 model on 200 semantically different tasks and a TSim-Instruct-300 model on 300 semantically similar tasks. The TDiff-Instruct-200 model has comparable ROUGE-L scores to the TSim-Instruct-300 model (25.9 vs 25.8 respectively) despite the TDiff-Instruct-200 using 1.5x less training data. This finding suggests that lowering semantic similarity could be an alternative to increasing the dataset size. We hope that our results could be used to assist when researchers collect new datasets for low-resource tasks and languages.

2 Related Work

2.1 Instruction Fine Tuning

The method of "instruction tuning" was introduced by Wei et al., 2021[4], which showed that large pretrained language models that are finetuned on natural language instructions have significantly improved generalization capabilities on a wide variety of NLP benchmarks. Extending this work, Chung et al., 2022[6] studied the effects on model performance on scaling up the model size and the number of training tasks. One surprising result from this paper is that the authors conclude that "increasing the number of finetuning tasks improves performance, although the majority of the improvement comes up to 282 tasks [out of 1,836]"[6]. We build on this previous observation and demonstrate that scaling up the number of tasks is more effective if the additional tasks are semantically diverse.

Similarly, Wang et al. 2022[1] showed a log-linear relationship between the number of training task types and the model's performance on held-out unseen test tasks. The authors conclude that decreasing the similarity of training tasks can be an alternative to scaling model sizes to achieve the same performance improvements. Although further analysis is not done on the effects of task similarity, the authors graciously contribute an open-sourced dataset known as Super-NaturalInstructions (Sup-NatInst)[1], which comprises 1616+ NLP tasks that contain enough examples and metadata for us to perform thorough studies on the effects of semantic similarity on instruction finetuning.

2.2 Semantic Similarity

This work uses the concept of semantic similarity to group tasks in the Sup-NatInst dataset. Vector semantics, which represents the meaning of text as a vector, are used in every NLP application related to meaning[7]. To assess the similarity of text, there have been a variety of approaches, but as with most NLP applications today, neural network models outperform most traditional methods with transformer-based models serving as the cutting edge in semantic similarity research[8]. Our research leverages transformer-based models to yield embeddings, which are then compared using cosine similarity between embeddings. Wang et al. 2016 [9] take a similar approach using LSTM's and cosine similarity to assess semantic similarity. Additionally, in a survey of similarity distances, Saif et al.[10] note that cosine similarity is one of the most widely used distance metrics in assessing semantic similarity.

3 Approach

We first develop an approach to calculating a measure of the semantic similarity between tasks and categories in the Sup-NatInst dataset[1], and second, create a few training datasets that contain tasks of varying levels of semantic similarity, finetune a T5 model[3] on each training set, and evaluate the models on the Sup-NatInst test set to see how variation in semantic similarity affects model performance on unseen tasks using the ROUGE-L[11] score.

To construct the semantic groupings of tasks, the Sup-NatInst training dataset was filtered to exclude tasks with less than a threshold number of instances. To ensure the dataset was balanced, only use a fixed number of instances per tasks in each experiment. We then create embeddings for each task by feeding a few natural language fields into a transformer-based embedding model. Next, we calculate the cosine similarity between each pair of embeddings creating a complete graph with edge weights representing similarity. Finally, we use a greedy algorithm (described below and in Appendix A.3) to select tasks that are either the most or least semantically similar.

Using those semantic groupings as individual experiments, we train a T5-small model on each experiment. Across multiple experiments, we also vary the number of tasks in the training set. Finally, each finetuned model is evaluated on the Sup-NatInst[1] test set using the ROUGE-L score.

3.1 Selecting Semantically-Similar Embeddings

We use semantic similarity to measure how similar the meanings of two tasks are in the Sup-NatInst dataset[1]. Embeddings for each task were created by feeding the task definition, categories, and positive and negative examples into the *all-mpnet-base-v2* model from the Sentence Transformers library[12].

Initially, the approach to selecting semantically similar and different groupings of tasks was pursued through clustering the task embeddings. This approach had a variety of challenges as noted in Appendix A.2, so a new approach was developed based on the cosine similarity of the embeddings. The cosine between two vectors has been used as a similarity measure in many NLP tasks[10]. A cosine similarity matrix was constructed by taking the cosine between each pair of embeddings.

The similarity matrix can be viewed as a complete graph where the tasks are nodes and the edges are cosine similarity scores. Finding subgraphs that are most semantically similar or different is equivalent to the maximum edge weight clique problem (MEWCP), which seeks "to find a clique that maximizes the sum of edge weights within the corresponding complete subgraph".[13] The MEWCP is NP-hard[13] and has limited existing implementations available on the internet. A brute-force implementation approach was found[14], but the graph sizes in this project were too large for the brute-force approach. Instead of using an existing approach to this problem, a simple greedy algorithm was devised to group the semantically similar and different embeddings. This greedy algorithm is discussed in detail in Appendix A.3. Put simply, to find the most similar embeddings, the algorithm initializes the selected group with the two embeddings that are most similar, computes their average and selects the embedding which was the closest similarity to the group average to add to the group. The algorithm then repeatedly 1) takes the group average, 2) calculates the similarity between the ungrouped embeddings and the group average, and 3) adds the task with the greatest similarity in step 2 to the group. These steps are repeated until the desired number of embeddings is reached. To find a semantically different grouping, you select the embeddings with the minimum similarity score.

Our project focused primarily on task embeddings, but we also performed some analysis on category embeddings. Each task was assigned a category in the datatset and calculating the category embeddings involved finding the mean of all task embeddings in a category. Then, the cosine similarity was calculated between the different category embeddings, and the greedy algorithm was applied to find category groupings.

3.1.1 Group Similarity Score

Based on the cosine similarity approach, the normalized sum of a given set of embeddings can be calculated by summing all of their pairwise cosine similarities (edge weights) and dividing by the number of pairs of embeddings (edges). The number of edges in a complete graph is $\binom{n}{2} = \frac{n(n-1)}{2}$ where n is the number of embeddings in the group. This normalized group similarity score is reported in our experimental results below.

4 Experiments

4.1 Models

Our base model is the Text-to-Text Transfer Transformer (T5) model, which is an encoder-decoder Transformer language model[3]. The encoder is made of blocks consisting of a self-attention layer and a small feed-forward network. The decoder has a similar structure to the encoder except that it includes a standard attention mechanism after each self-attention layer that attends to the output of the encoder using casual self-attention. The final decoder is fed into a dense layer with a softmax output. In our project, we use Huggingface's publicly available 60-million parameter "t5-small" version.

4.2 Data

The Sup-NatInst dataset[1] is a dataset of a variety of NLP categories of tasks (e.g. classification, extraction, infilling, sequence tagging, text rewriting) meant to help language models generalize on unseen tasks using only human-readable instructions. The Sup-NatInst dataset is made up of 76 task categories, 1616 tasks, and 5 million task instances. The training set consists of 756 English tasks, which we will filter down to 359 tasks from which we will draw our semantically similar and different groupings. An example of one task is shown in Figure 1.

Task Type	Cause Effect Classification
Task ID	task828_copa_cause_effect_classification
Definition	In this task your given two statements. You must judge whether the second sentence is the cause or effect of the first one. Label the instances as "cause" or "effect" based on your judgment. The sentences are separated by a newline character.
Positive Example	**Input**: The women met for coffee. They wanted to catch up with each other. **Output**: cause **Explanation**: The women met for coffee because they wanted to catch up with each other.
Negative Example	**Input**: My body cast a shadow over the grass. The sun was rising. **Output**: effect **Explanation**: The rising of the sun isn't an effect of casting a shadow over the grass.
Instance	**Input**: The woman tolerated her friend's difficult behavior. The woman knew her friend was going through a hard time. **Valid Output**: ["cause"]

Figure 1: An example of one of the tasks in the Sup-NatInst dataset, which consists of a task name (Task Type), definition, positive example, negative example, and a task instance.

Each task can have multiple instances and each instance can have multiple outputs, where each output corresponds to a single data example. Each task is also tagged with a specific category, which is not shown in Figure 1.

4.2.1 Balancing the dataset

The distribution of task instances across tasks in the dataset is not uniform. Figure 2 shows a histogram of the number of tasks with a given instance count.

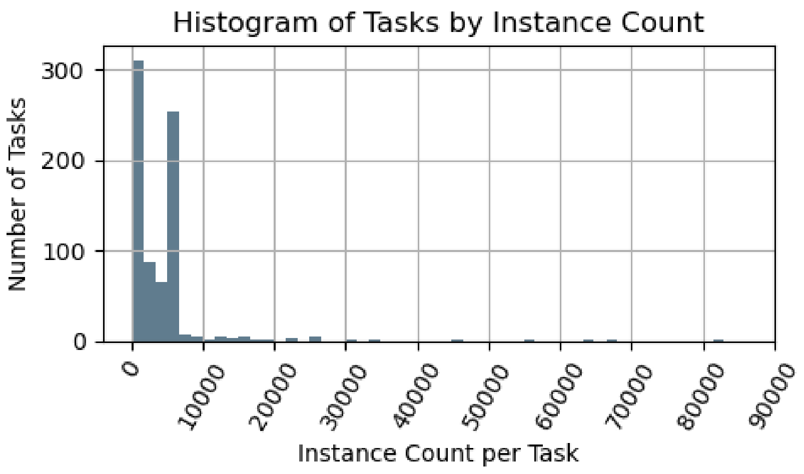

Figure 2: A histogram of the number of tasks by the count of instances in each task.

Figure 2 shows there are many tasks that have very few instances and a few tasks with a high number of instances. This non-uniform distribution creates a problem when constructing the groupings of semantically similar and different tasks where a single task won't dominate the grouping. To address this skewed distribution, we only select 3,250 instances from each task to ensure the distribution of instances per task is uniform in our finetuning datasets. This also means we filter out tasks with a total instance count of fewer than 3,250 instances, which is how we move from 759 tasks to 359 tasks as only 359 tasks have instance counts greater than 3,250. The value of 3,250 was chosen because it balanced the number of tasks available and the size of the total dataset.

4.3 Evaluation method

To ensure consistency with the Sup-NatInst paper, the Recall-Oriented Understudy for Gisting Evaluation: Longest Common Subsequence (ROUGE-L) score[11] will be the evaluation metric. We leveraged evaluation code from the Natural Instructions github repo[15] produced by the Allen Institute for AI, the authors of the Sup-NatInst paper. To evaluate our models, we will be using the Sup-NatInst test set[1]. Finally, the reported ROUGE-L for each model is calculated by averaging the individual ROUGE-L scores for each of the 117 tasks in the test dataset.

4.3.1 Baselines

We create a "Random" baseline model that is trained on a random selection of tasks from the 359 tasks in our filtered training set. This random baseline allows us to observe the performance for a given number of tasks without the effect of semantic similarity. This baseline is the most comparable baseline with our experiments because it has identical hyperparameters and the number of tasks seen during training.

While we include the 60M-parameter Tk-Instruct model[1] in our comparison table as a baseline, we also create another baseline by using the full Sup-NatInst dataset to train a 60M-parameter T5 model to understand what performance would be like with our chosen model and hyperparameters. We chose to have our own baseline using the full dataset due to the fact that replicating the ROUGE-L scores in the Tk-Instruct paper[1] was challenging within the time constraints of the project.

Finally, we also include the untrained 60M-parameter "T5-small" to show the generalization performance of the model without instruction finetuning.

4.3.2 Experimental details

We set up our experiments to vary the amount of semantic similarity among tasks in the training dataset. Specifically, we use the iterative cosine similarity algorithm described in subsection 3.1 on semantic similarity to group the most "similar" and most "different" tasks together to construct the training dataset for each model variant.

- TSim-Instruct models are trained on the **most similar tasks**.
- TDiff-Instruct models are trained on the **most different tasks**.

In each experiment, we vary the number of tasks included in the training dataset from 50, 100, 200, to 300 tasks, with each task containing 3,250 task instances for a total of 162.5K, 325K, 650K, and 975K training task instances. We use an AWS g5.2xlarge instance with an Nvidia A10G GPU with 24GB memory to train for 2 epochs using a batch size of 32 and a linearly decaying learning rate of 0.0002206 on the AdamW optimizer. The learning rate was selected by running a random hyperparameter search between the values 8e-4 and 1e-6, which we found are the ranges used by practitioners for finetuning T5. The maximum input and output token length is 512, and 128, respectively. Each training run takes between 3 to 12 hours, depending on the number of examples.

4.4 Results

Table 1 summarizes our results for the experiments which vary the degree of task similarity in each training set. Figure 3 shows that for all numbers of task types, the TDiff-Instruct models scored the highest and the TDiff-Instruct-300 achieved the best results with a ROUGE-L score of 26.6. We note that TDiff-Instruct-200 and TSim-Instruct-300 have comparable ROUGE-L scores (25.9 vs 25.8 respectively) even while TDiff-Instruct-200 used 1.5x less training data. We observe the trend that the difference in ROUGE-L score becomes smaller between the experiments as we increase the number of tasks. As we will discuss in the analysis section, this performance convergence in the larger-task models is related to the convergence in the semantic similarity between the two similar and different groupings.

Our best results come from the "Full dataset" model that is trained on all 756 tasks available, which achieved a ROUGE-L score of 30.0. This result is surprising since a similiar 60M-parameter Tk-Instruct[1] that is trained on the same number of tasks reported a ROUGE-L score of 40.1. It is likely that this gap is the result of the difference in hyperparameters and also the fact that Tk-Instruct is using a LM variant of the T5 model.

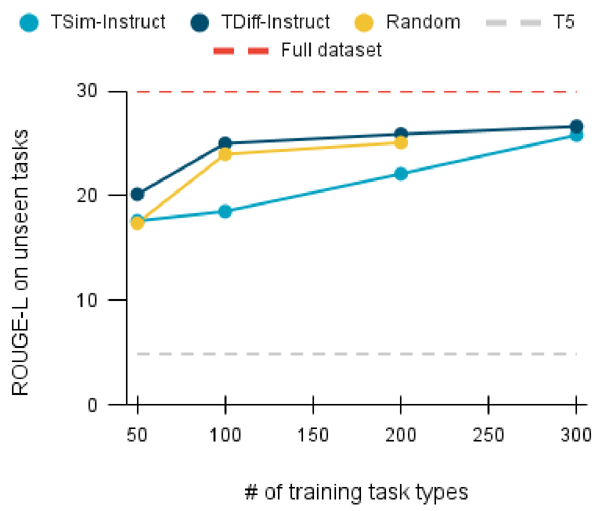

Figure 3: The model's ROUGE-L performance to unseen tasks as a function of the number of task types used in training. We observe that 1) increasing the number of training tasks and 2) lowering semantic similarity generally leads to better performance. The performance gap narrows as both datasets converge to the average dataset similarity.

Models	ROUGE-L score	Number of tasks seen
	Baselines	
T5-small	4.9	0
Random (ours)	25.1	200
Full dataset (ours)	**30.0**	756
Tk-Instruct [1]	40.1*	756
	Category grouping	
TSim-Category-Instruct	21.0	364
TDiff-Category-Instruct	**22.0**	233
	Task grouping	
TSim-Instruct-200	22.1	200
TSim-Instruct-300	25.8	300
TDiff-Instruct-200	25.9	200
TDiff-Instruct-300	**26.6**	300

Table 1: ROUGE-L scores on unseen tasks by models that are finetuned on different groupings of tasks. We show that T-Diff-Instruct outperformed Random, and all T-Sim-Instruct variants, even while using an equal or smaller number of training tasks. *We note that results for Tk-instruct are not directly comparable to our experimental setup.

5 Analysis

Figure 3 shows that **performance increases with more training tasks**, which is in agreement with previous research[4, 1, 16]. We also observed that **performance increases with lower semantic similarity**. Comparing the performance of the TDiff-Instruct, Random, and TSim-Instruct models in Figure 3 shows that the models with lower semantic similarity scores in Figure 4 have higher performance. The "Category groupings" in Table 1 also show this result of lower similarities in the training set improving the ROUGE-L score. However, the performance gains were more limited in the categories experiments due to challenges with the category embeddings.

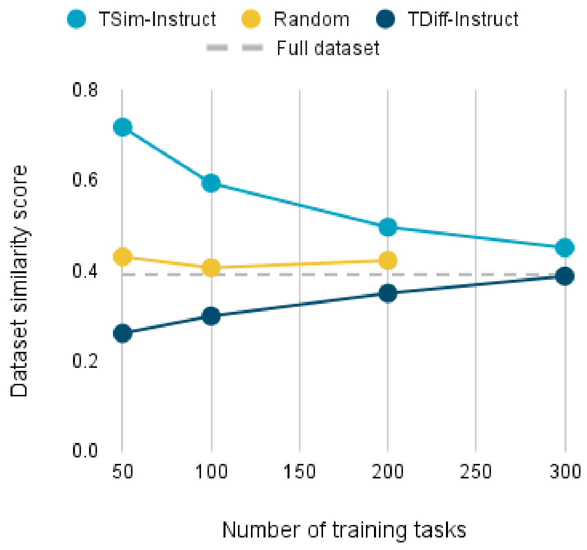

Figure 4: The cosine similarity of the dataset grouping is plotted as a function of the number of tasks. As the data subsets get larger, their semantic similarity trends toward the similarity of the full training dataset.

The performance of TSim-Instruct appears almost always below the random baseline, which indicates that the model overfits the data due to the training tasks being too similar. This conclusion is supported by the comparatively lower training loss for all TSim-Instruct models shown in Appendix 5. Figure 3 also shows that for 300 tasks, the performance gap vanished. This is because there were only 359 total tasks with instance counts above 3,250, so the similar and different groupings of 300 tasks were close to identical. This overlap in the 300 task similar and different datasets can be seen in the cosine similarity in Figure 4.

The relationship between the number of tasks and semantic similarity. When increasing the number of tasks in the TDiff-Instruct both the model performance and semantic similarity increase. This means the positive effect of increasing the number of task instances is overwhelming the negative effects of increasing the semantic similarity, suggesting that increasing training tasks has a stronger impact on performance than lowering semantic similarity. To put it in other words, training on more tasks will generally improve performance but the rate of increase in model performance by adding additional tasks depends on the semantic similarity of those additional tasks.

Increasing task diversity is an alternative to increasing the number of task types. In addition, the performance gap between the TDiff-Instruct and TSim-Instruct models in Figure 3 shows that lower similarity can be an alternative to increasing the number of task types to improve model performance. TDiff-Instruct which is trained on only 200 tasks has a comparable ROUGE-L performance as TSim-Instruct trained on 300 tasks. This means we are getting comparable performance using only 1.5 times less data. Additionally, TDiff-Instruct-300 shows a competitive ROUGE-L score within 11.3% of our best model while using 2.5x less training data. These results indicate that there could exist a reduction in training dataset size for models using instruction finetuning, depending on the task diversity represented by the training dataset and an acceptable performance regression threshold.

5.1 Limitations and Discussion

The main contribution that this work provides is to highlight the moderate boost that can be gained from using semantically different tasks and, more importantly, the more significant performance reductions that come from selecting semantically similar tasks. Our work suggests that research into building datasets for low-resource languages or tasks can use semantic similarity as a framework for assessing what additional tasks or examples to add to expand a dataset. This work relies on a deep-learning model to produce the embeddings, but future work could be done to compare these results with embedding produced from simpler embeddings based on term frequency-inverse document frequency or bag-of-word approaches. These simpler embeddings may be attractive for low-resource languages, which may not have deep learning embedding models trained in the native language.

This project also analyzed the effect of semantic similarity of categories on model performance. The models trained on semantically similar and different categories ended up having very similar ROUGE-L performances. Other than the processing step of averaging task embeddings to yield category embeddings, the same steps were performed to construct similar and different category groupings as the task embeddings. The category groupings also had quite different similarity scores of 0.49 and 0.71 for the different and similar groupings, respectively. One explanation for why the category semantic similarity groupings did not have much effect on ROUGE-L performance is that the averaging of the task embeddings to get the category embeddings corrupted the semantic meaning of the category embeddings. Using the cluster approach discussed in Appendix A.4 could validate alternative embeddings.

5.2 Future Work

Due to time limitations, this project only performed one training run for each experiment. If given more time, we would re-run each experiment three times to get an average and standard deviation for each measurement. This is especially important because the performance gap in Figure 3 between the random experiment and the semantically different experiment is fairly small and could be within a standard deviation of each measurement. If the uncertainty on the ROUGE-L measurement were larger than this gap, this finding would invalidate the conclusion that the semantically different grouping performed better than random selection.

The semantic separation of category embeddings didn't work as well for tasks. As an alternative approach, we could consider getting a definition of each natural language category and creating an embedding using only that definition to see if that embedding could more meaningfully separate the categories.

This work used a simple greedy algorithm to create semantically similar and different groupings. Using a more sophisticated solution to the Maximum Edge Weight Clique Problem would result in semantic groupings that have an even greater distance between them, potentially making the difference in performance more pronounced.

Lastly, other works ([1], [4]) have evaluated models on a few different evaluation measures other than ROUGE-L. It would be worthwhile to verify that the performance gaps observed in this work are also shown on a variety of NLP metrics.

6 Conclusion

In this work, we demonstrated that finetuning on tasks with greater semantic diversity (lower similarity) results in increased model performance. We also confirmed previous research showing improvements from using more tasks for IFT. We highlight how the positive improvements from increasing the number of tasks outweigh the negative effects of increasing the semantic similarity of the dataset. The impact of semantic similarity also holds for categories, though more research is needed to effectively assess category semantics. Finally, using our results we show that we can finetune a model on less data that's more semantically diverse and achieve the same level of zero-shot performance as another model that's trained on more data that is more semantically similar.

We hope our results could provide some guidance on how researchers could use semantic similarity to efficiently collect future datasets to address low-resource tasks and languages. We demonstrate that improving semantic diversity in a dataset can be an alternative to increasing the number of task types to achieve higher performance.

7 Acknowledgements

We would principally like to thank Siyan Li for being a great mentor. We also appreciated the advice Eric Frankel provided. A huge thanks to Chris Manning and the TA's for a great class!

References

[1] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, and et. al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks, 2022.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[3] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[4] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners, 2021.

[5] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[6] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models, 2022.

[7] Dan Jurafsky and James Martin. 2023.

[8] Dhivya Chandrasekaran and Vijay Mago. Evolution of semantic similarity—a survey. *ACM Computing Surveys*, 54(2):1–37, feb 2021.

[9] Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. Sentence similarity learning by lexical decomposition and composition. *CoRR*, abs/1602.07019, 2016.

[10] Saif M. Mohammad and Graeme Hirst. Distributional measures of semantic distance: A survey. *CoRR*, abs/1203.1858, 2012.

[11] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[12] Huggingface Inc. Huggingface inc, sentence tranformers, all-mpnet-base-v2. 2023.

[13] Seyedmohammadhossein Hosseinian, Dalila B. M. M. Fontes, Sergiy Butenko, Marco Buongiorno Nardelli, Marco Fornari, and Stefano Curtarolo. *The Maximum Edge Weight Clique Problem: Formulations and Solution Approaches*, pages 217–237. Springer International Publishing, Cham, 2017.

[14] Pierre Landais. Pierre-lds/graph-theory github repo, 2023.

[15] Allen Institute for AI. Natural instructions github repo, 2023.

[16] Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Natural instructions: Benchmarking generalization to new tasks from natural language instructions. *CoRR*, abs/2104.08773, 2021.

[17] Sci-Kit Learn. Clustering tutorial, 2023.

A Appendix

A.1 Code repository

The code used to analyze the data and train the models can be found in this repository: https://github.com/dzubke/cs224n-project.

A.2 Challenges with Clustering for Semantic Similarity

Clustering proved to be challenging to use in constructing semantically different groupings. Different approaches were done to select the optimal number of clusters, but these approaches never yielded a satisfying answer. Even if the number of clusters was selected, choosing the semantically similar tasks involved finding a cluster that had more tasks than the desired number in the grouping. Selecting the semantically different grouping proved even more challenging. Initially, randomly selecting on task from each cluster was taken, but this approach became harder when the number of clusters is less than the size of the desired grouping. It is also unclear if taking a clustering approach to creating semantically similar and different groupings would result in the optimal selection.

A.3 Greedy Approach to the Maximum Edge Weight Clique Problem

Since the Maximum Edge Weight Clique Problem (MEWCP) is NP-hard, the brute-force approach takes a very long time. Approximate methods for solving this problem exist[13], but no working implementation could be successfully installed within the scope of this project. Instead, a basic greedy algorithm was devised to create similar and different groupings. The pseudo-code for this algorithm is described below to find the most similar grouping:

```
1. Select the pair of embeddings with the max similarity
2. While the length of the grouping is less than the desired length
    (a) calculate the average embedding of the grouping
    (b) calculate the cosine similarity between the group average and all embeddings not in the group
    (c) add to the group the embedding with the maximum similarity with respect to the group average
3. return the grouping of the desired length
```

To find the semantically different grouping, the algorithm selected the minimum similarity in step (c).

A.4 Overlap between categories and task embedding clusters

To explore the meaning and validity of the task embeddings, a clustering of the task embeddings using k-means was performed to see how this clustering compared with the assigned categories in the dataset. There are 76 categories in the Sup-NatInst dataset, and when running the k-means clustering, the number of clusters was set to 76. Since there was no way to align individual k-means clusters with a specific category, a pair-wise confusion matrix was used to assess if the pairs of embeddings were clustered in the same way across the clusters. The description of the confusion matrix below is taken directly from the sci-kit learn documentation[17].]

$$\begin{bmatrix} C_{OO} & C_{O1} \\ C_{1O} & C_{11} \end{bmatrix}$$

For the confusion matrix above, each element can be interpreted as:

- C_{OO}: number of pairs with both clusterings having the samples not clustered together
- C_{1O}: number of pairs with the true label clustering having the samples clustered together but the other clustering not having the samples clustered together
- C_{O1}: number of pairs with the true label clustering not having the samples clustered together but the other clustering having the samples clustered together

- C_{11}: number of pairs with both clusterings having the samples clustered together

The actual confusion matrix calculated between the category cluster and k-means, task-embedding clusters is shown below.

$$\begin{bmatrix} 2,359,822 & 6,178 \\ 183,128 & 51,028 \end{bmatrix}$$

This means that 2,359,822 pairs were not both clusters did not have the samples clustered together, 6,178 pairs where the categories had the samples clustered together but the embedding clusters did not, 183,128 pairs where the embedding clusters had the pair in the same cluster but the categories did not, and 51,028 pairs where both clusterings had the pair together.

Using this confusion matrix, the F1-score can be calculated. The recall metric is 0.22 and the precision is 0.89. Thus, the F1-score is 0.35, which is fairly low.

We would assume that the tasks in the same category share semantic meaning, so the low overlap between clusters produced by the task embeddings suggests that the task embeddings may not be that semantically meaningful.

A.5 Training

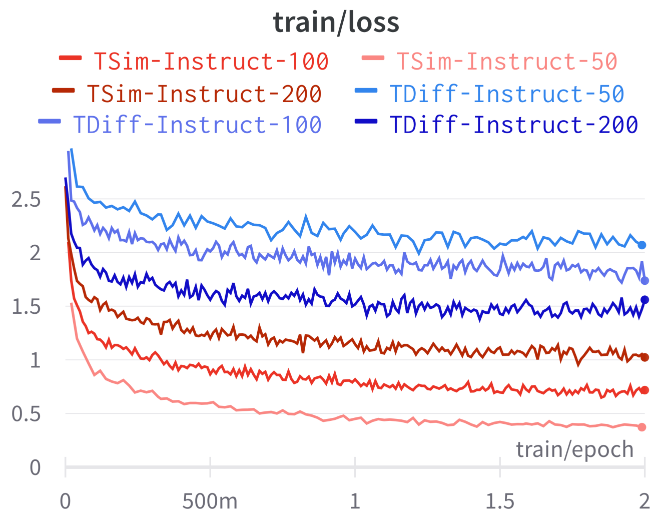

Figure 5: Training loss for TDiff-Instruct and TSim-Instruct models. TSim-Instruct models always showed a lower loss but lower ROUGE-L scores, suggesting possibility of overfitting.