Fantastically Ordered Prompts:
and Where to find them: Overcoming the high Few-Shot Prompt Order Sensitivity

Scalable Data Mining Term Project

Snehashri P G
Hardik Soni

Abstract and Background about Project

Large pretrained language models (LLMs) have demonstrated remarkable capabilities in various natural language processing (NLP) tasks. However, their performance often degrades when presented with only a few training examples, also known as the few-shot learning setting. This limitation necessitates the use of a large amount of labeled data for fine-tuning, which can be resource-intensive and time-consuming.

Intriguingly, studies have shown that LLMs, such as GPT-3, can achieve competitive results with fully-supervised, fine-tuned models even when provided with only a handful of training samples. This suggests that the order in which the training samples are presented can significantly impact the model's performance.

We investigate this phenomenon in detail and establish that:

- The order effect is present across model sizes, including the largest current models.
- The order effect is not limited to a specific subset of samples.
- A good permutation for one model is not transferable to another.

While using a development set to identify optimal permutations is possible, it deviates from the true few-shot setting due to the requirement for additional labeled data.

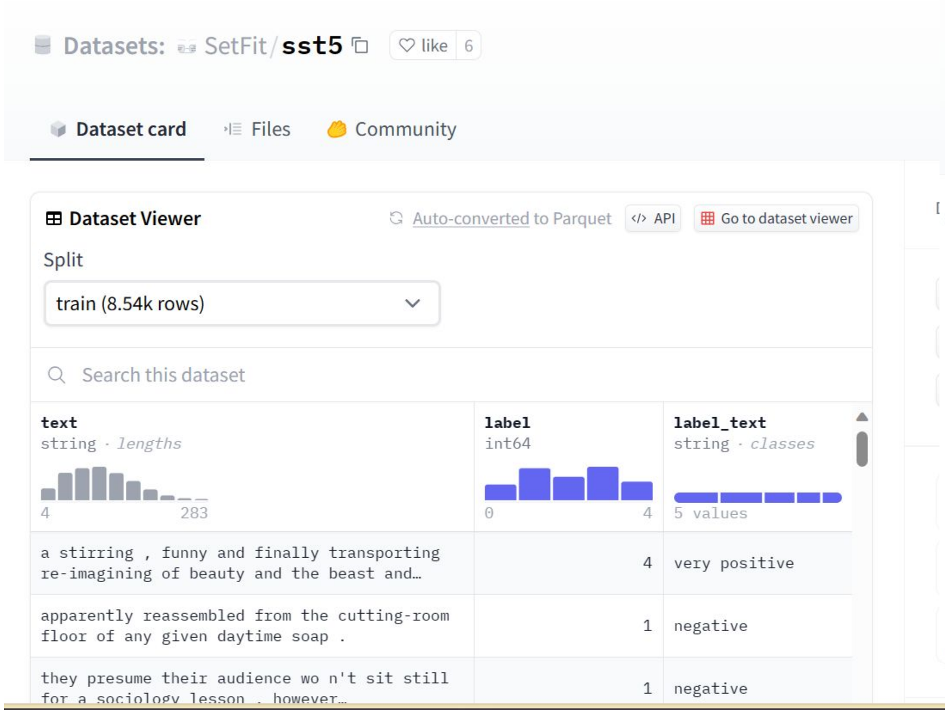
To address this challenge, we leverage the generative nature of LLMs to construct an artificial development set. By analyzing the entropy statistics of candidate permutations on this set, we identify effective prompts. Our method achieves a 13% relative improvement for GPT-family models across eleven established text classification tasks.

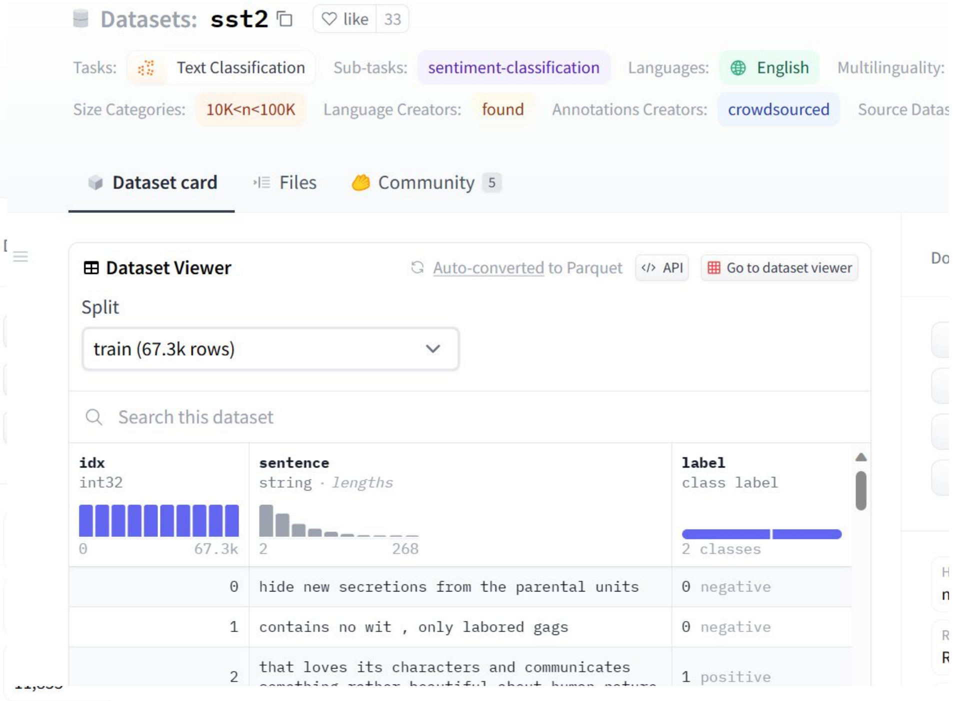
Datasets Used for Experiment

Dataset	# of Classes	Avg. Len.	Balanced
SST-2 (Socher et al., 2013)	2	12.4	Yes
SST-5 (Socher et al., 2013)	5	23.1	No
MR (Pang and Lee, 2005)	2	25.7	Yes
CR (Hu and Liu, 2004)	2	22.1	Yes
MPQA (Wiebe et al., 2005)	2	3.9	Yes
Subj (Pang and Lee, 2004)	2	28.9	Yes
TREC (Voorhees and Tice, 2000)	6	11.6	No
AGNews (Zhang et al., 2015)	4	53.8	Yes
DBPedia (Zhang et al., 2015)	14	65.5	Yes
CB (De Marneffe et al., 2019)	3	69.7/8.4	No
RTE (Dagan et al., 2005)	2	55.3/11.9	Yes

4-shot for most datasets, except DBPedia (1-shot), AGNews (2-shot)

Pretrained Datasets used in Experiment





Pretrained LLMs used in Experiment

Model card | Files | Community 68 | Train | Deploy | Use in Transformers

Edit model card

GPT-2

Test the whole generation capabilities here:

https://transformer.huggingface.co/doc/gpt2-large

Parameters

Model	Layers	Dim	Heads	Params
Small	12	768	12	186M
Medium	24	1024	16	437M
Large	24	1536	16	881M

Downloads last month
22,751,274

Safetensors | Model size | 137M params
Tensor type | F32

Space using TurkuNLP/gpt3-finnish-large 1

shri1510/TurkuNLP-gpt3-finnish-large

Prompt Construction and Examples

	Example
training set	(the greatest musicians, 1) (redundant concept, 0)
linearization	Review: the greatest musicians. Sentiment: positive Review: redundant concept. Sentiment: negative
concatenation	Review: the greatest musicians. Sentiment: positive. Review: redundant concept. Sentiment: negative *OR* Review: redundant concept. Sentiment: negative. Review: the greatest musicians. Sentiment: positive


```
SST-5:

{'text': 'a stirring , funny and finally transporting re-imagining of beauty and the beast and 1930s horror films', 'label': 4, 'label_text': 'very positive'}
{'text': 'apparently reassembled from the cutting-room floor of any given daytime soap .', 'label': 1, 'label_text': 'negative'}
{'text': "they presume their audience wo n't sit still for a sociology lesson , however entertainingly presented , so they trot out the conventional science-ficti
{'text': 'the entire movie is filled with deja vu moments .', 'label': 2, 'label_text': 'neutral'}
{'text': 'this is a visually stunning rumination on love , memory , history and the war between art and commerce .', 'label': 3, 'label_text': 'positive'}
```

```
SST-2:

{'idx': 0, 'sentence': 'hide new secretions from the parental units ', 'label': 0}
{'idx': 1, 'sentence': 'contains no wit , only labored gags ', 'label': 0}
{'idx': 2, 'sentence': 'that loves its characters and communicates something rather beautiful about human nature ', 'label': 1}
{'idx': 3, 'sentence': 'remains utterly satisfied to remain the same throughout ', 'label': 0}
{'idx': 4, 'sentence': 'on the worst revenge-of-the-nerds clichés the filmmakers could dredge up ', 'label': 0}
```

Libraries used in Experiment

datasets==2.1.0
jsonlines==3.0.0
omegaconf==2.1.1
openai==0.18.1
torch>=1.10.2
tqdm==4.62.3
transformers==4.16.2
wandb==0.13.5

Implementation plan

We employ four different versions of GPT-2 (Radford et al., 2019) with varying parameter sizes (0.1B, 0.3B, 0.8B, and 1.5B) and two versions of GPT-3 (Brown et al., 2020) with parameter sizes of 2.7B and 175B. Due to limitations in the context window size for the GPT-2 models (up to 1024 words), we use a 4-shot setting for all datasets except AGNews and DBPedia. We generate 24 different permutations for each set of randomly selected training samples and use five different sets (except for GPT-3 with 175B parameters, where we only use two sets with 12 permutations due to the high cost) for each experiment. This results in a total of 120 runs. We report the average and standard deviation of the evaluation metric across five different sets. To select effective prompts, we rank candidate prompts using the LocalE and GlobalE probing metrics on an automatically generated probing set. For Selection of the Performant Prompts we may devise multiple strategies some are explained in the next slide.

Loading LLM models:

```python
# Example using Hugging Face Transformers
from transformers import GPT2LMHeadModel, GPT2Tokenizer

model_name = "gpt2"
tokenizer = GPT2Tokenizer.from_pretrained(model_name)
model = GPT2LMHeadModel.from_pretrained(model_name)
```

```
vocab.json: 100% 1.04M/1.04M [00:00<00:00, 28.4MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 16.5MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 42.4MB/s]
config.json: 100% 665/665 [00:00<00:00, 23.9kB/s]
model.safetensors: 100% 548M/548M [00:05<00:00, 170MB/s]
generation_config.json: 100% 124/124 [00:00<00:00, 4.74kB/s]
```

Synthetic development set created using generative nature of LLM:

```python
generator = pipeline('text-generation', model='gpt2')
set_seed(42)
generator("Movies review:", max_length=18, num_return_sequences=1000)
```

[40] Python

```
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
{'generated_text': "Movies review:\n\nIf you'd like to get in touch, please email me"},
{'generated_text': 'Movies review: the perfect gift from the greatest of gifts to our son, Astr'},
{'generated_text': "Movies review: Watch 'Em All The Time\n\nAfter four years of waiting,"},
{'generated_text': 'Movies review:\n\n\n"In an unexpected turn to the \'90s, director'},
{'generated_text': 'Movies review: Watch Out For the Spider-Man 2: The Phantom Menace,'},
{'generated_text': 'Movies review:\n\nReview from New York Times:\n\nReview from Chicago Tribune'},
{'generated_text': 'Movies review: The End of Life by A.J. Lippmann (2005'},
{'generated_text': 'Movies review:\n\n"We loved the look, look and the characters, and'},
{'generated_text': 'Movies review: A little history and then some\n\n"As far as I\'m'},
{'generated_text': 'Movies review:\n\nThis is a film which has all the qualities of being a'},
{'generated_text': 'Movies review:\n\nThe movie I am looking for is "Pineapple Express'},
{'generated_text': 'Movies review: Bollywood actors who play tough, smart, tough.\n\n\n"'},
{'generated_text': 'Movies review: A Very Simple Movie Summary\n\nStarring: Daniel Radcliffe,'},
{'generated_text': "Movies review:\n\nThe '90s were filled with a lot of weird movies"},
{'generated_text': "Movies review: I couldn't get bored watching them.\n\nSo yeah. I"},
{'generated_text': 'Movies review: A Look at Film History\n\nThe British Columbia Movie Review publishes in'},
{'generated_text': 'Movies review: "S.Mighty Morphin Power Rangers: Force Awakens"\n'},
{'generated_text': 'Movies review:\n\nPretoria 2 (2004) (2008)\n\n\n'},
{'generated_text': "Movies review: Top 10 'Star Wars: Episode VIII's\n\n\nStar Wars"},
{'generated_text': 'Movies review: R. Kelly\n\nby\n\nThe first installment of Star Trek'},
```

Selection of Performant Prompts

We include code for the following 5 methods for example selection that we compare with in the paper:

1. **random**: random demonstration examples,
2. **max-entropy**: max-entropy active learning baseline, select the permutation set with maximum entropy.
3. **best-of-k**: best demonstration out of k random sets using a dev set, We then select the top k samples with the highest entropy values (k = 4 in our experiments) from the available 24 permutations to use as performant prompts.
4. **oracle**: iteratively picking the best demonstration example using a dev set,
5. **reordering**: reordering demonstration examples using the Global Entropy method by Lu et al., 2022.

Code: Base Strategy

```
class BaseStrategy:
    def __init__(self, conf: DictConfig):
        self.output_dir = conf.output_dir
        self.write_config(conf)

    def __call__(self, proc: BaseProcessor, model: GPT2Wrapper, shot: int, **kwargs):
        results = self.read_run(shot)
        if results is None:
            results = self.run_strategy(proc, model, shot, **kwargs)
            self.write_run(results, shot)
        else:
            logging.info(f"Found cache for shot={shot}, skipping run.")
        return results
```

Code: Greedy Strategy

```python
class GreedyStrategy(BaseStrategy):
    def run_strategy(
        self, proc: BaseProcessor, model: GPT2Wrapper, shot: int, evaluate: bool = True
    ):
        logger.info(f"{self.class_name} - shot={shot}")

        if shot == 0:
            train_indices = []
        else:
            # retrieve previous indices and make a copy
            prev = self(proc, model, shot - 1, evaluate=False)
            train_indices = prev["train_indices"][:]

            assert len(train_indices) + 1 == shot
            new_idx = self.acquisition(model, proc, train_indices)
            train_indices.append(new_idx)

        simple_result = {
            "shot": shot,
            "train_indices": train_indices,
        }

        if evaluate:
            prompts, cali_prompts = proc.create_prompts(train_indices)
            outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
            eval_result = proc.extract_predictions(outputs)
            self.write_result(eval_result, shot)
            self.write_train_examples(proc, train_indices, shot)
            simple_result["acc"] = eval_result["acc"]

        return simple_result
```

```python
class MaxEntropyStrategy(GreedyStrategy):
    def acquisition(
        self, model: GPT2Wrapper, proc: BaseProcessor, train_indices: List[int]
    ) -> int:
        # acquire the training example with the highest pred. entropy
        train_prompts, cali_train_prompts = proc.create_prompts(
            train_indices, split="train"
        )
        outputs = model.complete_all(
            train_prompts, calibration_prompts=cali_train_prompts
        )

        new_idx = max(range(len(outputs)), key=lambda i: entropy(outputs[i].probs))
        return new_idx


class OracleStrategy(GreedyStrategy):
    def __init__(self, conf: DictConfig):
        super().__init__(conf)
        self.max_train_dataset_size = 100
        self.test_subset = None

    def acquisition(
        self, model: GPT2Wrapper, proc: BaseProcessor, train_indices: List[int]
    ) -> int:
        # acquire the training example that results in the highest dev acc.
        # intractable to search everything, use small train & dev subsets
        best_idx = -1
        best_acc = float("-inf")
        for idx, example in tqdm(enumerate(proc.train_dataset)):
            eval_prompts, cali_prompts = proc.create_prompts(
                train_indices + [idx], split="val"
            )
            outputs = model.complete_all(eval_prompts, calibration_prompts=cali_prompts)
            eval_result = proc.extract_predictions(outputs, split="val")
            if eval_result["acc"] > best_acc:
                best_idx = idx
                best_acc = eval_result["acc"]

        return best_idx
```

```python
class GlobalEntropyOrderingStrategy(BaseStrategy):
    """Lu et al. - https://arxiv.org/pdf/2104.08786.pdf"""

    def run_strategy(self, proc: BaseProcessor, model: GPT2Wrapper, shot: int):
        logger.info(f"GlobalEntropyStrategy - shot={shot}")
        train_indices = random.sample(range(len(proc.train_dataset)), k=shot)
        # get all permutations
        perms = permutations(train_indices)

        probe_raws = []
        probe_examples = []
        for perm in permutations(train_indices):
            probe_raw = probe(model, proc.get_probing_prompt(perm))
            probe_str = probe_raw.strip().split("type:")[0]
            probe_item = proc.parse_probe_example(probe_str)
            probe_raws.append(probe_raw)
            probe_examples.append(probe_item)

        perm_to_entropy = {}
        for perm in permutations(train_indices):
            prompts, cali_prompts = proc.create_prompts(
                perm, split="custom", custom_split=probe_examples
            )
            outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
            eval_result = proc.extract_predictions(
                outputs, split="custom", custom_split=probe_examples
            )
            label_counts = torch.tensor(eval_result["class-dist"])
            class_distribution = label_counts / label_counts.sum()
            global_entropy = entropy(class_distribution)
            perm_to_entropy[perm] = global_entropy

        best_perm = max(perm_to_entropy.keys(), key=lambda k: perm_to_entropy[k])

        prompts, cali_prompts = proc.create_prompts(best_perm)
        outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
        eval_result = proc.extract_predictions(outputs)
        self.write_result(eval_result, shot)

        simple_result = {
            "shot": shot,
            "train_indices": best_perm,
            "acc": eval_result["acc"],
        }
        return simple_result
```

```python
class BestPermStrategy(BaseStrategy):
    def run_strategy(self, proc: BaseProcessor, model: GPT2Wrapper, shot: int):
        logger.info(f"BestPermStrategy - shot={shot}")

        perm_accs = []
        train_original = random.sample(range(len(proc.train_dataset)), k=shot)

        for train_indices in permutations(train_original):
            prompts, cali_prompts = proc.create_prompts(train_indices, split="val")
            outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
            eval_result = proc.extract_predictions(outputs, split="val")
            perm_accs.append((train_indices, eval_result["acc"]))

        best_indices = max(perm_accs, key=lambda t: t[1])[0]
        prompts, cali_prompts = proc.create_prompts(best_indices)
        outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
        eval_result = proc.extract_predictions(outputs)
        self.write_result(eval_result, shot)

        simple_result = {
            "shot": shot,
            "train_indices": best_indices,
            "acc": eval_result["acc"],
            "perm_accs": perm_accs,
        }
        return simple_result
```

```python
class BestOfKStrategy(BaseStrategy):
    def run_strategy(self, proc: BaseProcessor, model: GPT2Wrapper, shot: int):
        logger.info(f"BestOfKStrategy - shot={shot}")
        K = 10
        best_indices = None
        best_val_acc = -1.0

        for i in range(K):
            train_indices = random.sample(range(len(proc.train_dataset)), k=shot)
            prompts, cali_prompts = proc.create_prompts(train_indices, split="val")
            outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
            eval_result = proc.extract_predictions(outputs, split="val")
            if eval_result["acc"] > best_val_acc:
                best_val_acc = eval_result["acc"]
                best_indices = train_indices

        prompts, cali_prompts = proc.create_prompts(best_indices)
        outputs = model.complete_all(prompts, calibration_prompts=cali_prompts)
        eval_result = proc.extract_predictions(outputs)
        self.write_result(eval_result, shot)

        simple_result = {
            "shot": shot,
            "train_indices": best_indices,
            "acc": eval_result["acc"],
        }
        return simple_result
```

Results:

	SST-2	SST-5	DBPedia	MR	CR	MPQA	Subj	TREC	AGNews	RTE	CB
Majority	50.9	23.1	9.4	50.0	50.0	50.0	50.0	18.8	25.0	52.7	51.8
Finetuning (Full)	95.0	58.7	99.3	90.8	89.4	87.8	97.0	97.4	94.7	80.9	90.5
GPT-2 0.1B	$58.9_{7.8}$	$29.0_{4.9}$	$44.9_{9.7}$	$58.6_{7.6}$	$58.4_{6.4}$	$68.9_{7.1}$	$52.1_{0.7}$	$\mathbf{49.2}_{4.7}$	$50.8_{11.9}$	$49.7_{2.7}$	$50.1_{1.0}$
LocalE	$\mathbf{65.2}_{3.9}$	$34.4_{3.4}$	$53.3_{4.9}$	$66.0_{6.3}$	$\mathbf{65.0}_{3.4}$	$72.5_{6.0}$	$52.9_{1.3}$	$48.0_{3.9}$	$61.0_{5.9}$	$\mathbf{53.0}_{3.3}$	$49.9_{1.6}$
GlobalE	$63.8_{5.8}$	$\mathbf{35.8}_{2.0}$	$\mathbf{56.1}_{4.3}$	$\mathbf{66.4}_{5.8}$	$64.8_{2.7}$	$\mathbf{73.5}_{4.5}$	$\mathbf{53.0}_{1.3}$	$46.1_{3.7}$	$\mathbf{62.1}_{5.7}$	$\mathbf{53.0}_{3.0}$	$\mathbf{50.3}_{1.6}$
Oracle	$73.5_{1.7}$	$38.2_{4.0}$	$60.5_{4.2}$	$74.3_{4.9}$	$70.8_{4.4}$	$81.3_{2.5}$	$55.2_{1.7}$	$58.1_{4.3}$	$70.3_{2.8}$	$56.8_{2.0}$	$52.1_{1.3}$
GPT-2 0.3B	$61.0_{13.2}$	$25.9_{5.9}$	$51.7_{7.0}$	$54.2_{7.8}$	$56.7_{9.4}$	$54.5_{8.8}$	$54.4_{7.9}$	$52.6_{4.9}$	$47.7_{10.6}$	$48.8_{2.6}$	$50.2_{5.3}$
LocalE	$75.3_{4.6}$	$31.0_{3.4}$	$47.1_{3.7}$	$65.2_{6.6}$	$\mathbf{70.9}_{6.3}$	$67.6_{7.2}$	$\mathbf{66.7}_{9.3}$	$53.0_{3.9}$	$51.2_{7.3}$	$\mathbf{51.8}_{1.0}$	$47.1_{4.2}$
GlobalE	$\mathbf{78.7}_{5.2}$	$\mathbf{31.7}_{5.2}$	$\mathbf{58.3}_{5.4}$	$\mathbf{67.0}_{5.9}$	$70.7_{6.7}$	$\mathbf{68.3}_{6.9}$	$65.8_{10.1}$	$\mathbf{53.3}_{4.6}$	$\mathbf{59.6}_{7.2}$	$51.1_{1.9}$	$\mathbf{50.3}_{3.7}$
Oracle	$85.5_{4.3}$	$40.5_{6.3}$	$65.2_{7.6}$	$74.7_{6.1}$	$80.4_{5.4}$	$77.3_{2.3}$	$79.4_{2.4}$	$63.3_{2.9}$	$68.4_{8.0}$	$53.9_{1.3}$	$62.5_{7.4}$
GPT-2 0.8B	$74.5_{10.3}$	$34.7_{8.2}$	$55.0_{12.5}$	$64.6_{13.1}$	$70.9_{12.7}$	$65.5_{8.7}$	$56.4_{9.1}$	$56.5_{2.7}$	$62.2_{11.6}$	$53.2_{2.0}$	$38.8_{8.5}$
LocalE	$81.1_{5.5}$	$40.3_{4.7}$	$56.7_{7.5}$	$82.6_{4.2}$	$85.4_{3.8}$	$73.6_{4.8}$	$\mathbf{70.4}_{4.2}$	$56.2_{1.7}$	$62.7_{8.1}$	$53.3_{1.6}$	$38.4_{5.2}$
GlobalE	$\mathbf{84.8}_{4.1}$	$\mathbf{46.9}_{1.1}$	$\mathbf{67.7}_{3.6}$	$\mathbf{84.3}_{2.9}$	$\mathbf{86.7}_{2.5}$	$\mathbf{75.8}_{3.1}$	$68.6_{6.5}$	$\mathbf{57.2}_{2.3}$	$\mathbf{70.7}_{3.6}$	$\mathbf{53.5}_{1.5}$	$\mathbf{41.2}_{4.5}$
Oracle	$88.9_{1.8}$	$48.4_{0.7}$	$72.3_{3.3}$	$87.5_{1.1}$	$89.9_{0.9}$	$80.3_{4.9}$	$76.6_{4.1}$	$62.1_{1.5}$	$78.1_{1.3}$	$57.3_{1.0}$	$53.2_{5.3}$
GPT-2 1.5B	$66.8_{10.8}$	$41.7_{6.7}$	$82.6_{2.5}$	$59.1_{11.9}$	$56.9_{9.0}$	$73.9_{8.6}$	$59.7_{10.4}$	$53.1_{3.3}$	$77.6_{7.3}$	$55.0_{1.4}$	$53.8_{4.7}$
LocalE	$76.7_{8.2}$	$\mathbf{45.1}_{3.1}$	$83.8_{1.7}$	$78.1_{5.6}$	$71.8_{8.0}$	$78.5_{3.6}$	$69.7_{5.8}$	$53.6_{3.1}$	$79.3_{3.7}$	$\mathbf{56.8}_{1.1}$	$52.6_{3.9}$
GlobalE	$\mathbf{81.8}_{3.9}$	$43.5_{4.5}$	$\mathbf{83.9}_{1.8}$	$\mathbf{77.9}_{5.7}$	$\mathbf{73.4}_{6.0}$	$\mathbf{81.4}_{2.1}$	$\mathbf{70.9}_{6.0}$	$\mathbf{55.5}_{3.0}$	$\mathbf{83.9}_{1.2}$	$56.3_{1.2}$	$\mathbf{55.1}_{4.6}$
Oracle	$86.1_{1.5}$	$50.9_{1.0}$	$87.3_{1.5}$	$84.0_{2.7}$	$80.3_{3.3}$	$85.1_{1.4}$	$79.9_{5.7}$	$59.0_{2.3}$	$86.1_{0.7}$	$58.2_{0.6}$	$63.9_{4.3}$
GPT-3 2.7B	$78.0_{10.7}$	$35.3_{6.9}$	$81.1_{1.8}$	$68.0_{12.9}$	$76.8_{11.7}$	$66.5_{10.3}$	$49.1_{2.9}$	$55.3_{4.4}$	$72.9_{4.8}$	$48.6_{1.9}$	$50.4_{0.7}$
LocalE	$\mathbf{81.0}_{6.0}$	$42.3_{4.7}$	$80.3_{1.7}$	$75.6_{4.1}$	$79.0_{5.5}$	$72.5_{5.8}$	$54.2_{4.2}$	$54.0_{2.6}$	$72.3_{4.6}$	$50.4_{1.9}$	$50.5_{0.8}$
GlobalE	$80.2_{4.2}$	$\mathbf{43.2}_{4.3}$	$\mathbf{81.2}_{0.9}$	$\mathbf{76.1}_{3.8}$	$\mathbf{80.3}_{3.4}$	$\mathbf{73.0}_{4.3}$	$\mathbf{54.3}_{4.0}$	$\mathbf{56.7}_{2.0}$	$\mathbf{78.1}_{1.9}$	$\mathbf{51.3}_{1.8}$	$\mathbf{51.2}_{0.8}$
Oracle	$89.8_{0.7}$	$48.0_{1.1}$	$85.4_{1.6}$	$87.4_{0.9}$	$90.1_{0.7}$	$80.9_{1.4}$	$60.3_{10.3}$	$62.8_{4.2}$	$81.3_{2.9}$	$53.4_{3.1}$	$52.5_{1.4}$
GPT-3 175B	$\mathbf{93.9}_{0.6}$	$54.4_{2.5}$	$95.4_{0.9}$	$\mathbf{94.6}_{0.7}$	$91.0_{1.0}$	$83.2_{1.5}$	$71.2_{7.3}$	$72.1_{2.7}$	$85.1_{1.7}$	$70.8_{2.8}$	$75.1_{5.1}$
LocalE	$93.8_{0.5}$	$\mathbf{56.0}_{1.7}$	$95.5_{0.9}$	$94.5_{0.7}$	$91.3_{0.5}$	$\mathbf{83.3}_{1.7}$	$75.0_{4.6}$	$71.8_{3.2}$	$\mathbf{85.9}_{0.7}$	$\mathbf{71.9}_{1.4}$	$74.6_{4.2}$
GlobalE	$\mathbf{93.9}_{0.6}$	$53.2_{2.1}$	$\mathbf{95.7}_{0.7}$	$\mathbf{94.6}_{0.2}$	$\mathbf{91.7}_{0.4}$	$82.0_{0.8}$	$\mathbf{76.3}_{3.5}$	$\mathbf{73.6}_{2.5}$	$85.7_{1.0}$	$71.8_{1.9}$	$\mathbf{79.9}_{3.3}$
Oracle	$94.7_{0.2}$	58.2	$96.7_{0.2}$	$95.5_{0.2}$	$92.6_{0.4}$	$85.5_{0.8}$	$81.1_{4.9}$	$77.0_{1.2}$	$87.7_{0.6}$	$74.7_{0.4}$	$83.0_{0.9}$

Results:

ID	Template	Label Mapping
1	Review: {Sentence} Sentiment: {Label}	positive/negative
2	Input: {Sentence} Prediction: {Label}	positive/negative
3	Review: {Sentence} Sentiment: {Label}	good/bad
4	{Sentence} It was {Label}	good/bad

	Template 1	Template 2	Template 3	Template 4
GPT-2 0.1B	$58.9_{7.8}$	$57.5_{6.8}$	$58.1_{7.4}$	$56.6_{6.6}$
LocalE	$\mathbf{65.2}_{3.9}$	$\mathbf{60.7}_{4.6}$	$\mathbf{65.4}_{4.8}$	$61.0_{4.7}$
GlobalE	$63.8_{5.8}$	$59.0_{2.9}$	$64.3_{4.8}$	$\mathbf{63.5}_{4.8}$
GPT-2 0.3B	$61.0_{13.2}$	$63.9_{11.3}$	$68.3_{11.8}$	$59.2_{6.4}$
LocalE	$75.3_{4.6}$	$70.0_{7.2}$	$80.2_{4.2}$	$62.2_{3.4}$
GlobalE	$\mathbf{78.7}_{5.2}$	$\mathbf{73.3}_{4.5}$	$\mathbf{81.3}_{4.1}$	$\mathbf{62.8}_{4.3}$
GPT-2 0.8B	$74.5_{10.3}$	$66.6_{10.6}$	$70.3_{10.5}$	$63.7_{8.9}$
LocalE	$81.1_{5.5}$	$80.0_{5.6}$	$73.7_{6.2}$	$\mathbf{71.3}_{4.5}$
GlobalE	$\mathbf{84.8}_{4.1}$	$\mathbf{80.9}_{3.6}$	$\mathbf{79.8}_{3.9}$	$70.7_{5.3}$
GPT-2 1.5B	$66.8_{10.8}$	$80.4_{7.6}$	$54.5_{7.9}$	$69.1_{10.5}$
LocalE	$76.7_{8.2}$	$83.1_{3.6}$	$66.9_{7.5}$	$72.7_{5.5}$
GlobalE	$\mathbf{81.8}_{3.9}$	$\mathbf{83.4}_{3.2}$	$\mathbf{67.2}_{6.1}$	$\mathbf{74.2}_{5.3}$

Results:

	GPT-2 0.1B	GPT-2 0.3B	GPT-2 0.8B	GPT-2 1.5B
Baseline	$58.9_{7.8}$	$61.0_{13.2}$	$74.5_{10.3}$	$66.8_{10.8}$
LocalE	$\mathbf{65.2}_{3.9}$	$75.3_{4.6}$	$81.1_{5.5}$	$76.7_{8.2}$
GlobalE	$63.8_{5.8}$	$\mathbf{78.7}_{5.2}$	$\mathbf{84.8}_{4.1}$	$\mathbf{81.8}_{3.9}$
Split Training Set	$62.8_{5.3}$	$64.2_{6.1}$	$75.1_{6.8}$	$71.4_{7.8}$

Findings:

- Entropy based probing outperforms other techniques in performant prompt selection regardless of model size. It achieves around 13 percent relative improvement for GPT family models.
- Number of items in each sentiment affect the accuracy of LLM model and prompt should be carefully designed. Accuracy is higher when the classes are equal number of key-value pairs.
- Entropy based probing is effective on a family of LLM models and performs well on both models with smaller and larger number of parameters.
- Entropy-based In-context learning performs better than fine-tuned LLMs as observed and it is concluded that ordering of prompts in a way affects the performance of the model.
- Increasing model size of pretrained LLMs directs to increased accuracy but the computational complexity becomes higher.

References:

- **Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?** Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, Luke Zettlemoyer https://doi.org/10.48550/arXiv.2202.12837
- **An Explanation of In-context Learning as Implicit Bayesian Inference** Sang Michael Xie, Aditi Raghunathan, Percy Liang, Tengyu Ma https://doi.org/10.48550/arXiv.2111.02080
- **What Makes Good In-Context Examples for GPT-3?** Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, Weizhu Chen https://doi.org/10.48550/arXiv.2101.06804
- **Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity** Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, Pontus Stenetorp https://doi.org/10.48550/arXiv.2104.08786
- **Data Distributional Properties Drive Emergent In-Context Learning in Transformers** Stephanie C.Y. Chan, Adam Santoro, Andrew K. Lampinen, Jane X. Wang, Aaditya Singh, Pierre H. Richemond, Jay McClelland, Felix Hill https://doi.org/10.48550/arXiv.2205.05055
- **What Can Transformers Learn In-Context? A Case Study of Simple Function Classes** Shivam Garg, Dimitris Tsipras, Percy Liang, Gregory Valiant https://doi.org/10.48550/arXiv.2208.01066

Thank You