Kapa report

1. Use Case/Problem 1: Implementation of the Classifier

ML Problem Description and Potential Approaches :

Given a dataset of inputs, i.e. question-answer pairs, and their label among 6 categories, create a classifier that can map the inputs to their outputs well.

Generally speaking, there are two approaches to the this problem:

1. **Finetuning LLM-based models (FT)** to achieve the desired result: Given that we have 1000+ data points and 6 labels, this problem could be solved practically by leveraging the generalization abilities LLMs have as they have been trained on large amount of text and the perform language modelling pretty well.

   To perform transfer learning, we can simply optimize the full set of model parameters while replacing the classification head with a new randomly initialized one that specializes in our task of interest, i.e. predict one of the 6 categories.

   As an alternative, we could finetune the model without replacing its classification head by optimizing it to learn to predict the next work while performing instruction finetuning, ie. finetuning it on instruction-based prompts that involve the question-answer pairs and their labels in the prompt itself.

2. **Prompt-based zero/few shot in-context learning (ICL)** with text generative LLMs: text-generation LLMs have shown to perform incredible zero shot learning capabilities across many text-based tasks and therefore, we could experiment with performing in-context learning. In particular, we could instruct the model via prompting to generate the correct class of the question-answer pair without the requirement for any finetuning or model parameter optimization.

Research findings about 1 vs. 2:

- Data hunger: While ICL does require minimal to zero amount of data, FT needs a high-quality good amount of data for the optimization to generalize well to unseen examples. [1,2,3]
- Prompt engineering: While FT does not rely heavily on prompt formatting, ICL is shown to be quite sensitive to instructions and prompts that are essentially responsible for instructing the LLM to perform the task. In addition, it has been shown that the accuracy of prompt-based few-shot classification can vary to a large degree depending on the examples being provided in the prompts. [1,2]
- Model family:
    - FT makes use of encoder-only transformer architectures like BERT which is known for achieving SOTA performance on language understanding tasks like text classification as they have been trained to optimize for masked language modelling (MLM), leading to rich language representations that can be used across different downstream tasks. [1,2,3]
        - In addition, text generative LLMs, which have decoder-only architectures, like LLAMA and GPT families, can be finetuned in the same manner as Bert-like models by relying on the features extracted from the last hidden layer after decoding the last token in a sequence, followed by a new classification head. [1,2]

    - For zero/few shot learning, decoder-only LLMs are used as they have been trained for text generation and they excel at following instructions and predicting next relevant words. [1,2,3]
- Performance:
    - FT with encoder-only models (small models relative to decoder-only LLMs) still outperform all other methods in text classification across different tasks requiring good amount of data.

      As a rule of thumb, FT works well on 500+ examples and the more data the better. [1]
    - While applying FT to decoder-only LLMs is shown to achieve comparable good results to Bert-like models' performance, training decoder-only is usually very compute-expensive as they have 10x or 100x or 1000x parameters than Bert-like models. In that case, we can only use PEFT methods to optimize their parameters, which can be suboptimal or achieve results similar to encoder-only models at best. [1,2]
    - For zero/few shot ICL, only super large LMs can achieve comparable results to SOTA Bert-like models [1,2,3]. This makes sense as the larger the model the more powerful and generalizing it is.

      It has been shown that using the instruct-finetuned version of these models is way better than using the pretrained versions in both FT and ICL. [1,2]

Experiments:

Note: All FT experiments are shown on WandB here. Some specific information about each experiment is attached to the description of each experiment in the overview section!

Evaluation:

Since it is a classification task and we have an imbalanced dataset, I relied on calculating the F1-score with a 'weighted' mode to account for the importance of the class given its real representation in the dataset (regardless of train, val or test as they are have same class distribution through stratified splitting). This is motivated by the fact that the dataset class distribution represents real-world production case.

In addition to the F1-score, I log weighted recall, precision, the validation cross entropy loss and the per-label F1 score to WandB for further investigation.

FT with Bert-like models:

Given all the findings from literature, I went for experimenting with Bert-like models as we have good amount of data to make use and they are relatively small models so I can finetune their full set of parameters.

I experiments with several hyperparameters to tune to get best performance, namely learning rate, batch size, gradient clipping, weight decay, question-only vs. question+answer inputs, and dropout. In addition, I experimented with several architectures such as Bert, Bert multi-lingual, roberta, and electra.

I found the following hyperparameters as best configuration in this experiment achieving *weighted_F1=88% on the validation set*:

```
bsize=128
lr=0.00003
weight_decay=0.01
attention_probs_dropout_prob=0.1
classifier_dropout=null
hidden_dropout_prob=0.1
max_grad_norm=1
```

I observed that appending answers to the question inputs did not improve the performance at all in this experiment.

FT with decoder-only models:

I wanted to experiment with larger models to see if finetuning them would improve the previous results achieved by Bert. Therefore, I selected LLAMA 3-3B which is known to be trained on super large multi-lingual dataset and is able to understand code as well. I used LoRA for finetuning.

After tuning the same hyperparameter set above in addition to LoRA hparams (which matrices to decompose, rank, dropout, alpha) and weighted cross entropy, I found the following configuration as the best in this experiment achieving *weighted_F1=88% on the validation set*:

```
lora_alpha=16
lora_dropout=0.1
lora_r=32
lora_matrices=["q_proj", "v_proj"]
bsize=16
lr=0.0001
weight_decay=0.01
max_grad_norm=0.3
weighted_cross_entropy=False
```

I observed that appending answers to the question inputs did not improve the performance at all in this experiment.

As we see, FT with Bert and LLAMA could achieve similar results aligned with the research findings mentioned above. However, I assume that the more data we get the better these models can get especially in the case of Bert-like models.

Zero/few shot ICL:

From the research findings above, I found that using super large models can benefit ICL pretty much so I chose to go for one of the largest LLMs open-sourced which is instruct-LLAMA 3-70B. I used 4-bit quantization to be able to use it on an A100-80GB gpu for inference. For inputs, I use only questions without their ground truth answers as they turned out to be useless when finetuning Bert models.

From [1], we see that prompt design is really important for achieving good performance so I decided to implement three kinds of prompts to check which one works the best. I experimented with few shot ICL after I optimized the prompt design in zero-shot setting.

1. Generic prompt:

   ```
   Return the {target} {labels} with one word/label response without any additional text.
   Answer: {target_with_brackets}.
   ```

   where target is the target word to predict, i.e. in our case it is the type of question, labels is the list of labels among which the model shall choose, and the target_with brackets is just {target} so it understands that it should be one-worded answer, which enforces its formatting to be as brief as possible.

2. Generic prompt + task-specific info which is:

   ```
   """This question asked by the user is of the following typ
   es:
   - `Discovery`: User is exploring how to do something or lo
   oking something up.
   - `Troubleshooting`: User is asking about an error, often
   this is a stacktrace.
   - `Code`: The user is inputing code and asking kapa to cha
   nge, debug or explain something.
   - `Comparison`: User is asking kapa to contrast two things
   i.e. Are X and Y the same, what is the difference betwenn
   X and Y
   - `Advice`: The user is asking kapa what to do or what the
   best practice is.
   - `Off-Topic`: Anything else, could be something unrelate
   ```

```
d, gibberish, abuse and so on.
"""
```

3. 1+2+ more context for the model:

```
"This question is asked by the user to our support tech te
am and you should
classify it based on its type which represents the nature
of the issue the
user is encountering."
```

When performing zero shot ICL with prompt type. 1, I get on the validation set:

```
{'f1': 0.17286320590032295,
 'precision': 0.13487454778854008,
 'recall': 0.2565789473684211,
 'Advice': 0.25,
 'Code': 0.13333333333333333,
 'Comparison': 0.16666666666666666,
 'Discovery': 0.0,
 'Off-Topic': 0.2777777777777778,
 'Troubleshooting': 0.746268656716418}
```

When performing zero shot ICL with prompt type. 2, I get on the validation set:

```
{'f1': 0.8275137200685542,
'precision': 0.8454294091971503,
'recall': 0.8289473684210527,
'Advice': 0.5263157894736842,
'Code': 0.2,
'Comparison': 0.5,
'Discovery': 0.9398907103825137,
```

```
'Off-Topic': 0.6363636363636364,
'Troubleshooting': 0.8518518518518519}
```

When performing zero shot ICL with prompt type. 3, I get on the validation set:

```
{'f1': 0.8308177333464014,
'precision': 0.8461722391385704,
'recall': 0.8223684210526315,
'Advice': 0.6666666666666666,
'Code': 0.125,
'Comparison': 0.5,
'Discovery': 0.9273743016759777,
'Off-Topic': 0.6666666666666666,
'Troubleshooting': 0.8888888888888888}
```

Closely aligning with literature's findings, adding explicit task info and context about the task to the instruction prompt boosts the performance significantly, which is comparable to the SOTA bert performance.

Choosing prompt type 3. as our default prompt, I proceeded to the few shot ICL experimentation. By randomly selecting examples to be included in the prompt, I run 1-shot ICL 3 times and got various F1 scores on the validation set:

Run 1:

```
{'f1': 0.818100290285397,
'precision': 0.8498281729365859,
'recall': 0.8092105263157895,
'Advice': 0.6666666666666666,
'Code': 0.5,
'Comparison': 0.6666666666666666,
'Discovery': 0.8588235294117647,
'Off-Topic': 0.7407407407407407,
'Troubleshooting': 0.8771929824561403}
```

Run 2:

```
{'f1': 0.891545941874889,
'precision': 0.9174498746867169,
'recall': 0.881578947368421,
'Advice': 0.75,
'Code': 0.625,
'Comparison': 0.6,
'Discovery': 0.9318181818181818,
'Off-Topic': 0.9090909090909091,
'Troubleshooting': 0.9259259259259259}
```

Run 3:

```
{'f1': 0.8651968615436109,
'precision': 0.8707967032967033,
'recall': 0.8618421052631579,
'Advice': 0.42857142857142855,
'Code': 0.375,
'Comparison': 0.6153846153846154,
'Discovery': 0.9513513513513514,
'Off-Topic': 0.72,
'Troubleshooting': 0.9411764705882353}
```

As we see, few-shot ICL improved zero-shot ICL on average and achieves SOTA F1 scores compared to our Bert baseline.

However, by randomly selecting the examples per label to be introduced in the prompt, we do not have a robust approach for the choice of examples and we should run multiple runs hoping that we get a good set of examples with a high F1 validation score.

By running 2-shot ICL, I did not see drastic improvement of the F1 score compared to one-shot ICL but still, the average was slightly higher, indicating that including 2 examples per label is preferred in general. On the other hand, including >2

examples did decrease the performance to around 65% which is not a good score.

Based on these observations, I relied on 2-shot ICL as our default best ICL method for further experimentation. From [1], it seems that the choice of examples in the prompt influences the performance drastically. Therefore, to generalize better and have a better strategy for choosing examples, I have experimented with another idea which is as follows:

- Given a new input, i.e. question from a user, we can retrieve from the training set the most similar question to that input across all labels based on Bert-embedding similarities. Then, we can introduce these questions with their labels, which are most matching to our input question, in the instruction prompt and perform few-shot ICL.

When experimenting with this method of example selection and doing several runs, I found that the variance of the F1 validation score across runs is very low which means, this method is way more reliable and robust than random selection, to be able to evaluate few shot ICL more reliably. I got on average 87.5% of F1 validation score across runs.

Another heuristics to try out for example selection is here.

Takeaway

Taking all results together, we can see that these findings are very reliable as they align with research findings about LLM-based text classification. The solution to this problem has different approaches to follow but since we have 1000+ data points, we should reply on the full finetuning of relatively small Bert-like models as we can get more robust task-specific SOTA classifiers.

2. Use Case/Problem 2: Classification for Custom Labels

For production we should solve the classification problem in general as follows:

1. In case we have 1000+ data points, which is similar to the use case 1 above, we should reply on the full finetuning of relatively small Bert-like models as we can get more robust task-specific SOTA classifiers.
2. In case we have less data, the results of FT can become less reliable and therefore, we should rely on few-shot ICL. The experiments I did show that few-shot ICL works very well and achieves SOTA results if prompt design and the choice of LLM is done properly.

In particular, we should rely on super large models like GPT-4o or LLAMA-3-70B and use a prompt design that includes the task information, context about the classification problem and the same format as proposed above. Also, we should follow the heuristics I used for choosing the examples to be introduced in the prompt, which makes the text generation process more relevant to the question to be classified and more stable for achieving high accuracy.

Production setup: Based on the previous 2 cases, I think, we could be flexible in implementing this feature such that we make it optional for the customer.

- Either they provide a good amount of labelled data (question-custom category pairs) and in this case we could deploy a finetuning service that automatically finetunes a Bert model on their custom labels and deploys the classifier for that customer.
- Or they could choose to go for no-data approach and in that case we can deploy, alongside the finetuning service, a text generative LLM like LLAMA 3-70B, which is deployed to do inference and perform few shot ICL based on their defined labels.

Compute budget and resources should be taken into consideration here as well, given that I do not have access to such information yet.