Homework 1

11-766 Spring 2026

Due date: 02/05/2026 11:59 PM EST

In this homework, you will (1) explore how choice of prompt and evaluation method can influence one's conclusions about how well a model can do a test; (2) experiment with different methods for building a classifier using a limited amount of labeled data; and (3) analyze the properties of embedding and retrieval systems.

For Problems 1 and 2, we have provided starter code in the form of IPython notebooks. These IPYthon notebooks are designed to be run in Google Colab and may need modifications to run in local environments. Note that you will need a GPU machine to complete this homework–the TAs were able to complete all questions using a Colab Pro instance. If you have any trouble running the starter code or accessing a GPU machine, please post on Piazza or ask for help in office hours.

You must submit:

- Your revised versions of the two IPython notebooks. Though we are asking for your code, we do not plan to directly assess it (except to check for cheating). You may feel free to make any changes that support you in answering the questions in this document.
- Your written report as `report.pdf`, containing answers to the questions in the blue "Deliverables" boxes. When answering questions, please strive to be precise but succinct with your language.

As a reminder, you are permitted to use AI for the homework, so long as (1) all words that appear in your report are written by you unless you are explicitly quoting from an LLM's output; and (2) you include an "AI Attestation" section in your report that describes how you used AI to help you with the homework.

Problem 1: Evaluating Machine Translation

In class, we discussed how machine translation, the ability to automatically translate text from one natural language to another, was one of the key applications which motivated innovation in neural language modeling. In this question, you will use this application to explore topics in prompt engineering and system evaluation.

[Question 1.1] (***Standard Evaluation Methods for Machine Translation***) We have provided you a starter codebase which loads in the first 100 examples of the OPUS Books dataset[1] and uses the HuggingFace `SmolLM2-1.7B-Instruct` to translate the input French texts into English. We have chosen this model because it is small enough to run locally on most machines or in a Colab notebook. We have also provided implementations for three evaluation metrics: BLEU[2], COMET[3], and BERTScore[4]. Research these methods to answer the following questions.

DELIVERABLES FOR Q1.1

Please answer the following questions.

A. What are the main differences between BLEU, COMET and BERTScore? How does the mechanism by which each metric calculates "translation quality" differ across these three approaches?

B. Describe three distinct scenarios where these metrics might provide divergent or contradictory signals (e.g., one metric reports a high score while another reports a low one). For each case, support your argument with constructed French→English translation examples and provide technical justification for why this lack of alignment occurs.

C. For each of the three metrics, identify a specific application or project requirement where that metric would be the most appropriate choice over the others. Explain your answers.

[1] OPUS Books contains aligned sentences from public-domain fiction books.

[2] Bleu: a Method for Automatic Evaluation of Machine Translation (Papineni et al., ACL 2002)

[3] COMET: A Neural Framework for MT Evaluation (Rei et al., EMNLP 2020)

[4] BERTScore: Evaluating Text Generation with BERT (Zhang et al., ICLR 2020)

[Question 1.2] (***AI as Judge Evaluation***) In recent years, it has become popular to evaluate language generation systems by asking a strong language model to judge the outputs of the system. Design and implement an evaluation method that uses an LLM to judge the quality of the machine translation system's responses. You will need to select a judge model and design a prompt.

DELIVERABLES FOR Q1.2

Please answer the following questions.

A. Describe your AI judge method. You should include enough detail here that a classmate would be able to re-implement your method.

B. Compare and contrast your AI-as-judge method with one of the three evaluation methods from the previous question. Are their quality scores correlated? What differences do you observe in their biases and and errors?

[Question 1.3] (***Prompt Brittleness***) As we've discussed in class (and you may have discovered from the previous question), LLM performance can be extremely susceptible to choice of prompt. Even, semantically identical prompts that vary only in minor formatting can have significant different performance. In the starter code you used for the previous questions, we provided you a very simple prompt.

```
Translate this text from French to English. French text: <french text>
```

This prompt can be found as the `SYSTEM_PROMPT` in `llm_task1.ipynb`

We would like to experiment with modifying this prompt to try and find prompts that would have the same meaning to a human reader but result in very different performance characteristics.

DELIVERABLES FOR Q1.3

Please answer the following questions.

A. Identify two prompts that have different phrasings of the instruction of how to do the task (one of these can be the provided instruction) and result in a statistically significant difference in model performance. Demonstrate the difference with evaluations using the metrics from Questions 1.1-1.2 and examples.

B. Identify two prompts that have the same instruction but vary only in minor formatting (e.g. use of whitespace, capitalization, colons, etc.) and result in a statistically significant difference in model performance. Demonstrate the difference with evaluations using the metrics from Questions 1.1-1.2 and examples.

[Question 1.4] (***Back translation***) Back-translation is a common technique for evaluating the quality of a translator or a machine translation system. The core idea is to translate the text into a target language and then translate it back into the original source language. If the resulting back-translated text closely matches the original input, this suggests that the translation process has preserved the underlying meaning.

Note: In this experiment, we translate text from English to French and then back-translate it to English.

DELIVERABLES FOR Q1.4

Please answer the following questions.

A. Perform a qualitative analysis of the original English text compared to the back-translated English text. What kinds of errors stick out?

B. Visualize the correlation between machine translation quality according to the methods from 1.1/1.2 and similarity between the original and back-translated texts. Discuss whether the automatic metrics of machine translation quality are consistent with your observations of back-translation quality.

C. Besides machine translation, describe another application where the back-translation technique may be useful.

Problem 2: Fine-Tune Large Language Models

Suppose you are developing an app that gives personalized restaurant recommendations based on whether the restaurant makes foods you'll like. As part of the app, you'd like to build a classifier that predicts food preferences. Your final classifier should input a food name and output a preference score of 1 (wouldn't want to eat), 2 (ambivalent toward this food), or 3 (sounds very tasty).

You will explore an API-based classifier (using the OpenAI API) as well as finetuning a much smaller BERT model for the task. Since each of you are working with a different personalized dataset, we are not interested in your overall validation or test set accuracy. Rather, you will be graded on how well you demonstrate that you've conducted a well-reasoned exploration of the development of a classifier for this task.

[Question 2.1] (***Data annotation***) As a first step, take some time to annotate the 150 foods in this spreadsheet with your food preferences. You will need to make a copy of the spreadsheet.

This homework question will only be interesting if you have *some* food preferences. If you happen to be an unpicky omnivore who likes all foods, please imagine a dietary restriction for yourself (e.g. consider pretending that you can't stand raw vegetables or that all desserts are offensive). Similarly, if you are a supertaster who hates nearly all foods, you may consider inventing some food preferences. Optionally, you may define that three preferences labels in any way that makes sense to you; for example the omnivore among you may choose that 1 (tasty), 2 (extra tasty), and 3 (let me eat this right now!) is a better labeling scheme. The goal is to have a non-negligible representation of each label class.

DELIVERABLES FOR Q2.1

Please answer before you try running any classifiers:

A. What three class labels did you use, and how many examples does your dataset have for each class?

B. What accuracy would you expect from random guessing?

C. Do you believe that your food preferences will be easy to classify? Take a guess at what accuracy you expect your best classifier to achieve, and explain your answer.

[Question 2.2] (***OpenAI inference***) Build the best classifier you can using the OpenAI API. We have provided a simple few-shot prompt that you can use as a starting point. For full marks, you must demonstrate you have made a reasonable effort to explore inference techniques and API feature that goes beyond this starting implementation. In the IPython Notebook, we've provided some suggestions for directions to explore. Please try to spend no more than $15 USD on experimentation, as you may need the rest of your OpenAI credit for the homeworks and project.

DELIVERABLES FOR Q2.2

Please answer:

A. Share with us a log of the configurations you tried and the validation set accuracy they achieved. In your log, explain your thought process behind trying out these configurations. For each configuration, report validation set accuracy.

B. Which configuration did you decide was best? What is the test set accuracy of this system?

[Question 2.3] (***Finetuning***) Use the provided starter code to finetune BERT[5] to perform the classification tasks. Try to improve upon the base training configuration provided in the starter code. You may change any hyperparameters, modify or expand upon the training data, or try out a different model. However, you should stick to models that are under 200 million parameters.

DELIVERABLES FOR Q2.3

Please answer:

A. Share with us a log of the configurations you tried and the validation set accuracy they achieved. In your log, explain your thought process behind trying out these configurations. For each configuration, report validation set accuracy.

B. According to the definition of language model provided in the first week of class, would you consider BERT to be a language model? Explain your answer.

C. Suppose you expect your new app to have about 1,000 users. For each user, you expect do do about 1,000 food classifications per year (using that user's customized classifier). Would you choose the API approach from Part 2.2, or the finetuning approach from Part 2.3, or some other method entirely? Justify your answer—consider traits such as performance, cost, latency, and efficiency.

[5] "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding " (Devlin et al., NAACL 2019)

Problem 3: Retrieval

[Question 3.1] (***Limits of embedding-based retrieval***) Let $\mathcal{D} = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n\}$ be a corpus of variable-length strings over alphabet Σ, and let q be a query string. Consider two retrieval approaches:

1. **Embedding approach**: Map each string and the query through an embedding function $f : \Sigma^* \to \Re^{d_{\text{emb}}}$ into a d_{emb}-dimensional vector space. Retrieve the top-m strings by selecting those with smallest distance $d(f(q), f(\mathbf{s}))$ in embedding space, where d is a distance metric (e.g., Euclidean or cosine distance).

   Example: Given query $q =$ "cardiac arrest", an embedding model might retrieve strings containing semantically similar phrases like "heart attack" or "myocardial infarction" even if they don't contain the exact query terms.

2. **Regex approach**: Translate the query q into a regular expression $r(q)$ and retrieve all exact matches from the corpus: $\{\mathbf{s} \in \mathcal{D} : \mathbf{s} \in \mathcal{L}(r(q))\}$, where $\mathcal{L}(r(q))$ denotes the formal language (set of all strings) accepted by the regular expression $r(q)$.

   Example: Given query $q =$ "error code", translate to regex $r(q) =$ ".*error[_\s]code[_\s]?\d+.*" to retrieve strings matching patterns like "error code 404" or "error_code_500" with precise pattern matching.

For a given query q, assume there exists a ground-truth set of relevant strings $C^* \subseteq \mathcal{D}$ (which may be defined by human annotators or some oracle). Let $\tilde{C}$ denote the set of strings retrieved by a given approach. Define retrieval quality using standard information retrieval metrics:

$$\text{P} = \frac{|\tilde{C} \cap C^*|}{|\tilde{C}|} \quad \text{(fraction of retrieved strings that are relevant)}$$

$$\text{R} = \frac{|\tilde{C} \cap C^*|}{|C^*|} \quad \text{(fraction of relevant strings that are retrieved)}$$

When comparing two retrieval approaches, consider precision and recall separately[6] (i.e., one approach may have higher precision even if it has lower recall, and vice versa). Assume that both f and r are constructed using the training data. Additionally, assume that the choice of the cutoff m (for top-m retrieval) is fixed and is part of this construction.

DELIVERABLES FOR Q3.1

Please answer the following questions.

A. Characterize the formal conditions under which regex-based retrieval can achieve higher *precision* than embedding-based retrieval, and separately, conditions under which it can achieve higher *recall*. Your answer should:

- Identify properties of the relevant set C^*, the corpus $\mathcal{D}$, or the query q that favor regex approaches
- Express conditions as precisely as possible using mathematical notation
- Provide concrete examples illustrating each condition
- Briefly explain what about embeddings prevents them from achieving the same precision/recall as regex (as appropriate)

B. Conversely, characterize conditions under which embedding-based retrieval can achieve higher *precision* than regex-based retrieval, and separately, conditions under which it can achieve higher *recall*. Discuss limitations of regex-based retrieval relative to C^*.

[6] In practice, it is common to summarize the precision–recall tradeoff using a single scalar metric such as the F1 score (at a chosen operating point, dependent on both P and R) or Average Precision (AP) / area under the precision–recall curve (PR-AUC) (as a threshold-free summary). Which summary metric is most appropriate depends on the application.

[Question 3.2] (*Limits of kNN-LM*) Question: When Does kNN-LM Interpolation Hurt Performance?

Assume we have a datastore of prefixes and associated next token. More formally, given a embedding function f that maps sequences of tokens to a d_{emb}-dimensional space, we have a datastore, $\mathcal{D} = \{\langle f(w_1, \ldots, w_{n-1}), w_n \rangle\}$ built from a large corpus of sentences. Assume further that we have a retrieval function that, given a prefix $w_{<n} = w_1, \ldots, w_{n-1}$, retrieves the set of k closest elements of $\mathcal{D}$; we will use $\mathcal{D}_k(w_{<n})$ to refer to this set.

A kNN language model [1] is defined as,

$$\Pr_{\text{kNN}}(w|w_{<n}) \propto \sum_{\langle f(w'_{<n'}), w'_{n'} \rangle \in \mathcal{D}_k(w_{<n})} \mathrm{I}[w = w'_{n'}] \exp\left(-d\left(f(w'_{<n'}), f(w_{<n})\right)\right)$$

where $\mathrm{I}[x]$ returns 1 if the logical expression x is true and 0 otherwise; $d(x, y)$ returns the distance between two vectors $x, y \in \Re^{d_{\text{emb}}}$. Given a base language model $\Pr_{\text{LM}}(w|w_{<n})$, we define the interpolated kNN-LM as,

$$\Pr(w|w_{<n}) = \lambda \Pr_{\text{kNN}}(w|w_{<n}) + (1 - \lambda) \Pr_{\text{LM}}(w|w_{<n})$$

where $\lambda \in [0, 1]$ is an interpolation parameter.

Consider the case where we evaluate a language model with a separate test corpus C and macro-averaged perplexity,

$$\text{PPL}(C, \Pr(\cdot)) = \frac{1}{|C|} \sum_{\mathbf{s} \in C} \left(\prod_{i=1}^{|\mathbf{s}|} \Pr(s_i|s_{<i}) \right)^{-\frac{1}{|s|}}$$

DELIVERABLES FOR Q3.2

Please answer the following questions.

A. Derive conditions under which the interpolated model $\Pr(w|w_{<n})$ achieves *worse perplexity* than the base language model $\Pr_{\text{LM}}(w|w_{<n})$ on C. Your analysis should:

- Identify specific properties of the corpus $\mathcal{D}$, the retrieval set $\mathcal{D}_k(w_{<n})$, or the prefix $w_{<n}$ that lead to degradation
- Express these conditions as precisely as possible (using mathematical notation where appropriate)
- Provide brief intuitive explanations for why each condition causes degradation

Your answer should be formal and precise, but need not be a complete mathematical proof. Aim for a half page of clear, concise analysis.

B. Based on your analysis in part A, discuss how the conditions under which kNN-LM fails relate to when retrieval-augmented generation (RAG) might be ineffective more generally. Be specific about which insights transfer and which might not.

References

[1] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations (ICLR)*, 2020.