

# Assignment 4

CMSC 473/673 — Introduction to Natural Language Processing

Due Wednesday December 1st, 11:59 PM

| Item     | Summary                                                          |
|----------|------------------------------------------------------------------|
| Assigned | Friday November 12th                                             |
| Due      | Wednesday December 1st                                           |
| Topic    | Probabilistic Language Modeling and Noisy Channel Classification |
| Points   | 100                                                              |

In this assignment you will implement and evaluate probabilistic language models. You will use them to build noisy channel classifiers.

You are to *complete* this assignment on your own: that is, the code and writeup you submit must be entirely your own. However, you may discuss the assignment at a high level with other students or on the discussion board. Note at the top of your assignment who you discussed this with or what resources you used (beyond course staff, any course materials, or public Piazza discussions).

The following table gives the overall point breakdown for this assignment.

| Question | 1  | 2  | 3  | 4  |
|----------|----|----|----|----|
| Points   | 25 | 15 | 30 | 30 |

You have incredible freedom in how you structure your code. Your code does not have to be pristine. However, we should be able to follow what you write. Ensure your code is appropriately commented and described in your writeup. Try to make the coding easy on yourself. The recursive definitions permit rather simple methods for constructing higher-order  $N$ -gram models.

**What To Turn In** Turn in a writeup in PDF format that answer the questions; turn in all requested code necessary to replicate your results. Be sure to include specific instructions on how to build (compile) your code. Answers to the following questions should be long-form. Provide any necessary analyses and discussion of your results.

**How To Submit** Submit the assignment on the submission site:

<https://www.csee.umbc.edu/courses/undergraduate/473/f21/submit>.

Select “Assignment 4.”

## Full Questions

1. **(25 points)** Read Chapter 4 (except Section 4.9) of 3SLP (<https://web.stanford.edu/~jurafsky/slp3/4.pdf>). Write a summary (at least 1/2 a page, but likely more) of Naive Bayes classifiers. In your summary, be sure to address the following:
  - What are the motivations/assumptions of a Naive Bayes classifier?
  - How does it use Bayes rule?
  - The text around equation 4.6 mentions “features.” How are these “features” related to (i) the features (especially bag-of-words) features we discussed earlier in the semester, and (ii) count-based language models?
  - How would you train and use an  $n$ -gram-based Naive Bayes classifier? Formulate your answer both mathematically and in prose.
2. **(15 points)** Assume you have a corpus with two sentences:
  - the cat chased the fluffy mouse
  - the cat chased the squeaky mouse

Notice that this corpus provides 6 types. To have a robust  $N$ -gram language model, we must add in two additional *output types*: [EOS] and [UNK]. In class, we also talked about how we can optionally add an additional special symbol, [BOS] to handle the initial seeding of the model. Let’s consider a bigram model,  $p(z|y)$ .

- (a) How many different distributions  $p(z|y)$  must you learn? As a hint, one of them is  $p(z|y = \text{cat})$ . Identify all values of  $y$  that give rise to a different distribution.
  - (b) For each valid combination of  $z$  and  $y$ , compute an add-1 bigram estimate. Write out each value  $p_{\lambda=1}(z|y)$ , where  $p_{\lambda=1}(z|y) \propto \text{count}(y, z) + 1$ .
  - (c) Using your answer from part (b), compute the perplexity of the sentence “the mouse chased the kitty”.
3. **(30 points)** This question has you implement and evaluate six different language models that apply at the sentence level. These implementations and scripts developed here are needed for subsequent questions.

To receive full credit on *this* problem, you must

- decide on a Universal Dependencies dataset to use (see **Dataset**).
- implement (and turn in your code for) the six models described below (see **Models**).
- briefly discuss, in your writeup, the code layout/design choices made.
- discuss how you handled the OOV, BOS, and EOS symbols (including any hyperparameters that could affect their use and the proper inclusion or exclusion from computing the probability distributions).
- give a detailed discussion in your writeup of how you verified your implementations are correct (this can be concise).
- produce and include a file that replicates the perplexity results from Q2. That is, train your laplace-1 model on the sentences in Q2, replicate the perplexity values from that question, and include the output of those results as a file in your submission).

- show that you can train a model and compute dev perplexity results by completing the following table (see **Perplexity**). Your perplexity values must be finite (except for the mle), but you do not need to obtain any particular perplexity value.

| Basic Model Type | N-gram | Hyperparameters      | Dev perplexity |
|------------------|--------|----------------------|----------------|
| mle              | 1      | ???                  | ???            |
|                  | 2      | ???                  | ???            |
| laplace          | 1      | $\gamma = ???, ???$  | ???            |
|                  | 2      | $\gamma = ???, ???$  | ???            |
| ar_lm            | n/a    | hyperparams=???, ??? | ???            |
|                  | n/a    | hyperparams=???, ??? | ???            |

“ar\_lm” stands for “autoregressive language model,” which is just a general term that encompasses recurrent language models.

**Models** Implement the following six language models: (i) a unigram maximum likelihood (unsmoothed) model, (ii) a bigram maximum likelihood (unsmoothed) model, (iii) a unigram Laplace model, (iv) a bigram Laplace model, (v) an autoregressive neural network (with one configuration of hyperparameters), (vi) a autoregressive neural network (with a second configuration of hyperparameters),

Your code must be able to handle  $N \in \{1, 2\}$ , i.e., unigram and bigram models.

- (a) (model name: **mle**)  $N$ -gram models with no smoothing,

$$p^{(n)}(x_i | x_{i-n+1}, \dots, x_{i-1}) \propto c(x_{i-n+1}, \dots, x_i). \quad (1)$$

- (b) (model name: **laplace**)  $N$ -gram models with Laplace smoothing, with hyperparameter  $\gamma > 0$ ,

$$p_{\gamma}^{(n)}(x_i | x_{i-n+1}, \dots, x_{i-1}) \propto c(x_{i-n+1}, \dots, x_i) + \gamma. \quad (2)$$

- (c) (model name: **ar\_lm**) A recurrent neural language model, of your choice. I recommend choosing among an Elman RNN (torch.nn.RNN), an LSTM (torch.nn.LSTM), or a GRU (torch.nn.GRU). However, you may choose to use a Transformer model if you wish, or another recurrent network.

You may use Pytorch, Tensorflow, or Keras to implement the ar\_lm models. *Hint: what happens to the laplace model if  $\gamma = 0$ ? Use this to simplify your implementation!*

**Dataset** You must experiment with at least one language, of your choice, from the Universal Dependency dataset. You may experiment with multiple languages, but you are not required to do so (and not doing so will not penalize you). Regardless of the language(s), **model each sentence separately**. That is, prepend (if you wish) each sentence with BOS and append EOS to each sentence.

**Perplexity** You will evaluate each language model  $p$  using perplexity,

$$\text{ppl}(p) = \exp \left( \frac{-1}{M} \sum_{i=1}^M \log_e p(w_i | h_i) \right). \quad (3)$$

The logarithm here is assumed to be the natural logarithm. The sum iterates over history-observation instances  $h_i, w_i$ ; depending on the order of the model under consideration, the history  $h_i$  may actually be a sequence of words. There are  $M$  of these instances. This is the “token” view, i.e., we process each observed word (given its history) in turn.

4. (30 points) In this final question, you will implement a language identification system: given a sentence, identify (from a small [2-3 size] set of languages) the likely language that sentence is written in. You will develop and evaluate your system on languages from the Universal Dependencies dataset.

- **Your Task** Your system will be based upon the *noisy channel* model of classification: given a sentence  $s$  tokenized into  $S$  words  $w_1, \dots, w_S$ , you will predict which language  $l$  that  $s$  is most likely to be in according to the language with the highest posterior probability:

$$\hat{l} = \arg \max_l p(l \mid w_1, \dots, w_S) \quad (4)$$

$$= \arg \max_l \overbrace{p(w_1, \dots, w_S \mid l)}^{\text{likelihood of } l \text{ generating } s} \underbrace{p(l)}_{\text{prior probability of language } l} \quad (5)$$

$$= \arg \max_l \log p(w_1, \dots, w_S \mid l) + \log p(l). \quad (6)$$

Here,  $p(w_1, \dots, w_S \mid l)$  is a language-conditioned probabilistic language model, and will be discussed more below.

Create and turn in a report in which you:

- Document internal progress and development by training on the training sets and evaluating on the development sets for each language you use. As in the previous question, this internal development **must** include some amount of experimentation.
- Discuss why the value  $l$  that optimizes (6) is equivalent to the one that optimizes (4).
- Evaluate the best configuration on the testing set, and compare against a baseline language ID model.
- Present the results, both internal and final, in readable formats, e.g., in tables or plots, and clearly indicate in the writeup what model and model configurations resulted in the best test set performance.
- Summarize and analyze your results: what did you learn, and what do your experiments and results teach us about language ID?
- Describe, and justify, any design decisions or assumptions you made along the way.
- Discuss and document any tests or verifications you performed to ensure your models form proper probability distributions.

Turn in all code, including what is necessary for producing the plots.

- **How You Will Be Graded** Your grade on this problem will be from proper documentation of your code, experiments, and reporting of results on both the development and test portions.
- **System Specifics** The following elaborates on specifics for the system you are to build.

**Language Models** The likelihood of a sentence being generated by any language  $l$  is given by a language-conditioned probabilistic language model:  $p(w_1, \dots, w_S \mid l)$ . You may use any of the language models from the previous question.

**Prior Probabilities** The prior probability  $p(l)$  represents the probability of a language occurring at all, without regard to any particular input sentence. Often,  $p(l)$  is just the overall frequency of language  $l$  observed in training: e.g., if 100 sentences were in language A and 50 were in language B, then  $p(l = A) = \frac{100}{100+50} = \frac{2}{3}$ . Of course, the prior probability can itself be a smoothed estimate.

**Languages to Experiment on** Overall, you must experiment with at least three different languages: you can do this either with two different pairs of languages (e.g., first pair of A and B, and a second pair of C and D); two different pairs of languages, with one language shared (first pair of A and B, and a second pair of A and C); or one set of three (or more) languages (e.g., three-way classification of A, B, and C). While you may experiment on fully distinct languages, e.g. English vs. Korean, you may also experiment on different subsets of a particular language. For example, how easy is it to distinguish UD\_English-EWT from UD\_English-GUM?

**Proper Baseline Comparison** Include a proper baseline comparison. For this, you may implement the “most common” baseline. You can easily implement this by approximating the posterior via the prior probability:

$$\hat{l} = \arg \max_l p(l \mid w_1, \dots, w_S) \approx \arg \max_l p(l). \quad (7)$$

- **Evaluation** Evaluate your predictions using precision, recall, and F1 score. While you may implement these yourself, you may also use 3rd party libraries. For instance, the Python library `sklearn` has functions

`sklearn.metrics.precision_score` and `sklearn.metrics.recall_score`.

The first argument to each function is a vector of ground truth labels (the actual labels); the second argument is a vector of predicted labels. By setting the parameter `average = 'micro'` or `average = 'macro'`, you can compute micro vs. macro scores. For example, the following snippet first computes micro recall, then macro precision:

```
>>> gold = ['lang1', 'lang2', 'lang1', 'lang3']
>>> pred = ['lang3', 'lang2', 'lang1', 'lang3']
>>> sklearn.metrics.recall_score(gold, pred,
                                average = "micro")
0.75

>>> sklearn.metrics.precision_score(gold, pred,
                                    average = "macro")
0.8333333333333333
```

Notice that the labels can be strings.