CP1: Unigram Probabilities

Last modified: 2023-02-10 00:15

Status: **RELEASED.**

What to turn in:

- ZIP file of source code: https://www.gradescope.com/courses/496674/assignments/2582948/
- PDF report file: https://www.gradescope.com/courses/496674/assignments/2656376

Your ZIP file should include

- All starter code .py files (with your edits) (in the top-level directory)

Your PDF should include (in order):

- Your full name
- Collaboration statement
- Problem 1 figure 1a and short answers 1b and 1c
- Problem 2 figure 2a and short answer 2b

Questions?: Post to the cp1 topic on the discussion forums.

Jump to: Problem 1 Problem 2 Starter Code

Background

Recall the unigram model discussed in class and in HW1

In this CP, we'll use this model to analyze the text observed in the annual State of the Union speech by the US President, with a dataset going back to 1945.

- Train on speeches from 1945 - 2015 (almost 400,000 words!)
- Test on the speech in 2016

How well can we predict what was said in 2016?

Throughout, we'll assume that we have a known vocabulary with V distinct words, all known in advance. We'll indicate each of the V possible vocabulary words with an integer in $\{1, 2, \ldots V-1, V\}$.

Goals

The two problems below will address two key questions:

- Given training data, how should we estimate the probability of each word? How might estimates change if we have very little (or abundant) data?
- How can we select hyperparameter values to improve our predictions on heldout data, using only the training set?

Probabilistic Model

Likelihood model for one word

Let discrete random variable X denote an observable vocabulary word. X can realize any of the integer values $\{1, 2, \ldots V\}$.

By this definition, X will follow a *categorical* distribution, whose parameter is the vector variable $\mu = [\mu_1, \mu_2, \ldots \mu_V]$. The PMF of X is:

$$p(X = v|\mu) = \mu_v, \quad \forall v \in \{1, \ldots V\}$$

To define a valid PMF, the vector μ must have V non-negative entries and sum to one. These constraints are formalized in math here:

$$\sum_{v=1}^{V} \mu_v = 1$$

$$\mu_1 \geq 0, \mu_2 \geq 0, \ldots \mu_V \geq 0$$

Likelihood model for many words

We can observe a total list of N words as training data, $x_1, x_2, \ldots x_N$. Each x_n stands for an integer index to our vocabulary $x_n \in \{1, 2, \ldots V\}$.

We model our list of words as the realizations of N random variables, $X_1, \ldots X_N$, each one **identically distributed** according the categorical model above and **conditionally independent** of the other words given parameter μ.

In math, we can write the joint likelihood of the whole dataset as:

$$p(X_1 = x_1, X_2 = x_2, \ldots, X_N = x_N|\mu) = \prod_{n=1}^{N} p(X_n = x_n|\mu)$$
$$= \sum_{v=1}^{V} \mu_v^{n_v}$$

where n_v is a summary of the observed words $x_1, \ldots x_N$ indicating how many times term v appears in our list:

$$n_v = \sum_{n=1}^{N} [x_n = v]$$

Remember, the bracket expression is 1 if the expression inside is true, and 0 otherwise. This is commonly called Iverson bracket notation: https://en.wikipedia.org/wiki/Iverson_bracket

Prior model

Before seeing any data, we can assume the vector μ is drawn from a symmetric Dirichlet with scalar concentration parameter $\alpha > 0$.

$$p(\mu) = \text{DirPDF}(\mu_1, \ldots \mu_V; \alpha, \ldots \alpha)$$

Recall that this is intuitively like describing our beliefs about μ in terms of "pseudo-counts". That is, we act as if we have observed each vocabulary term α times before seeing any training data.

Experiment 1: How to estimate the probability of heldout words?

Given N observations, how can we estimate μ and use that to predict the next word?

Below, we provide the exact formulas for 3 common estimators for next word prediction.

Your task in Problem 1 (below) will be to implement these estimators and apply them to the provided training/test data.

Throughout all the estimators below, it is useful to view n_v as a function of the training data: $n_v(x_1, \ldots x_N)$. We will simply write n_v to avoid verbose notation, but keep in mind we determine the count n_v by what we observe in our training data.

Maximum Likelihood (ML) estimator

To model an (unseen) new word X_*, we can simply assume it is **conditionally independent** of other words given μ and **identically distributed** using the categorical model above.

$$p(X_* = v|\mu = \mu^{\text{ML}}) = \mu_v^{\text{ML}}, \qquad \mu_v^{\text{ML}} = \frac{n_v}{N}$$

Problem: If some word v never appears in the training set ($n_v = 0$), this estimator assumes it can never occur as the next word either. This is usually an un-usable model, because in real data analysis we will often know that some words could appear later even if we never see them in our training set (esp. for small train set sizes).

Maximum Likelihood (ML) estimator with reserved probability for unseen words

We can fix the problem above in an ad-hoc way by assuming that we *always* reserve $\epsilon > 0$ probability for unseen words in our vocabulary. This probability is uniformly distributed over all U unseen words, and the remaining $1 - \epsilon$ probably is distributed to each seen word according to the ML-estimate of μ:

$$p(X_* = v|\mu^{\text{ML}}) = \begin{cases} (1-\epsilon)\frac{n_v}{N} & \text{if } n_v > 0 \\ \epsilon \frac{1}{U} & otherwise \end{cases}$$

Again, the integer U is the total number of vocabulary words that have zero count: $U = \sum_v [n_v = 0]$.

If $U = 0$, we return to the plain ML estimator above, as it will not have problems when all words in the vocabulary are observed in the training set.

Maximum A-Posteriori (MAP) estimator

The ML estimator above did not take a **probabilistic** approach to handling how to estimate μ for vocabulary words. We just applied a post-hoc correction. Here, we develop an alternative that uses the full joint model defined above (both prior and likelihood).

To model a new word X_*, we first form the joint model over all random variables as:

$$p(X_*, X_1, \ldots X_N, \mu) = \text{DirPDF}(\mu; \alpha) \cdot \prod_{n=1}^{N} \text{CatPMF}(X_n|\mu) \cdot \text{CatPMF}(X_*|\mu)$$

Conditioning only on the N observed values in the training set, we can form the posterior over μ:

$$p(\mu|X_1 = x_1, \ldots X_N = x_N) = \text{DirPDF}(\mu|)$$

We can then obtain the point estimate of vector μ that maximizes the posterior's PDF function, by solving this optimization problem:

$$\mu^{\text{MAP}} \leftarrow \arg \max_{\mu \in \Delta^V} p(\mu|x_1, \ldots x_N)$$

This is called MAP (maximum a-posteriori) point estimation. We're finding the single value of vector μ that maximizes the posterior PDF.

Turns out, because of the conjugacy of the Dir-Cat model and the known properties of Dirichlet distributions, this estimator has a closed-form solution. Each entry in μ can be set to:

$$\mu_v^{\text{MAP}} = \frac{n_v + \alpha - 1}{N + V(\alpha - 1)} \quad \text{for } v \in \{1, 2, \ldots V\}$$

We can now plug-in this estimate to make next-word predictions!

$$p(X_* = v|\mu = \mu^{\text{MAP}}) = \frac{n_v + \alpha - 1}{N + V(\alpha - 1)}$$

Note that this estimator **requires** that $\alpha > 1$ unless every vocabulary word is observed at least once (otherwise the computed probability will not be positive and thus invalid).

Posterior Predictive estimator

The above strategies for prediction do not take into account any uncertainty about μ. They compute one specific μ vector, then plug this one value into the likelihood $p(X_*|\mu)$ to make predictions.

Instead, we consider *averaging over* our uncertainty about μ to make next-word predictions:

$$p(X_* = v|X_1 = x_1, \ldots X_N = x_N) = \int_{\mu \in \Delta^V} p(X_* = v, \mu|x_1, \ldots x_N) d\mu$$

As we showed in HW1, this integral has a closed-form due to Dir-Cat model's conjugacy:

$$p(X_* = v|X_1 = x_1, \ldots X_N = x_N) = \frac{n_v + \alpha}{N + V\alpha}$$

Experiment 2: How to select the hyperparameter settings using training data?

Naturally, the performance of the MAP and PPE estimators above is rather sensitive to the chosen value of the prior hyperparameter $\alpha > 0$. How can we select α in a principled way?

One useful way is via maximizing the marginal likelihood of the training set.

In Problem 2 below, you'll be asked to compute the **marginal likelihood** of the observed training words. This is also called the **evidence**.

As derived in class and in HW1, the marginal likelihood's PMF is:

$$\begin{aligned} p(X_1 = x_1, \ldots X_N = x_n) &= \int_{\mu \in \Delta^V} p(X_1 = x_1, \ldots X_N = x_n|\mu) p(\mu) d\mu \\ &= \frac{\Gamma(V\alpha) \prod_{v=1}^{V} \Gamma(n_v + \alpha)}{\Gamma(N + V\alpha) \prod_{v=1}^{V} \Gamma(\alpha)} \end{aligned}$$

Remember, this formula is specialized to a *symmetric* Dirichlet prior with scalar hyperparameter $\alpha > 0$. Every vocabulary term has the same "pseudocount" of α under this prior.

Hint 1 : Use `gammaln` for numerical stability

We suggest computing the *log* of the marginal likelihood PMF function directly (use SciPy's `gammaln` function as demonstrated in class). This will be more numerically stable, because of it works by adding in log space rather than multiplying in probability space where underflow or overflow are likely.

Hint 2 : How to compute a model's "score"

We also suggest that when you report a model's *score*, that is, the log likelihoods it assigns to any dataset (e.g. train set/test set), that you divide by the number of words. We call this "per-word" normalization, as follows:

$$\text{average-score-per-word}(x_1, \ldots x_N) = \frac{1}{N} \sum_{n=1}^{N} \log p(X_n = x_n)$$

This has two benefits:

- 1: these "scores" are easier to talk about (your answer should be roughly between -11 and -6, not a much larger negative number).
- 2: scores are comparable across datasets, despite different sizes

Starter Code + Data

You can find the starter code and all needed data in the course Github repository here:

https://github.com/tufts-ml-courses/cs136-23s-assignments/tree/main/unit1_CP

The data is provided inside the folder `state_of_union_1945-2016/`, you will find one plain-text file per speech:

- 1945-Truman.txt
- 1946-Truman.txt
- ...
- 2016-Obama.txt

We've provided all the necessary code for parsing this corpus into a Vocabulary and a list of training and test words.

See `run_demo.py` for an accessible example of how to use this data and our Estimator API.

The README file on github gives you a complete list of all the code implementation tasks you have (also listed below).

We'll assume you have already installed the course conda environment.

Problem 1

Tasks for Code Implementation

1(i): Implement `fit` and `predict_proba` methods of starter code MLEstimator.py

1(ii): Implement `fit` and `predict_proba` methods of starter code MAPEstimator.py

1(iii): Implement `fit` and `predict_proba` methods of starter code PosteriorPredictiveEstimator.py

Tasks for Report PDF

1a: Figure for Experiment 1: Test-set Log Likelihood vs Train Set Size

In your report PDF, using the starter code of `run_estimator_comparison.py`, produce 1 figure showing three overlapping line plots, one for each of the estimators you implemented above in (i) - (iii). Each estimator's line should show the estimated per-word log probability of the *entire* test data on the y-axis, as a function of the amount of available training data on the x-axis.

You should be sure to enforce the following settings:

- Set ϵ (`epsilon_unseen_proba`) = 0.00001 for the maximum likelihood estimator
- Set α (`alpha`) = 1.001 for both estimators that require using the Dirichlet prior
- Set frac_train_list = [1./128, 1./64, 1./32, 1./16, 1./8, 1./4, 1./2, 1.0]
- Do not change the plotting limits or tick labels (the starter code defaults are ideal)
- Report and plot the log probabilities using the per-word normalization already implemented in the `score` methods for each estimator released in the starter code (described above in Hint 2)

In your report PDF, provide a few complete sentences to each prompt below:

- **1b: SHORT ANSWER** Will the ML estimator eventually be clearly superior to the others, given enough training data? Why or why not? (Hint: what happens to formulas above as $N \rightarrow \infty$).

- **1c: SHORT ANSWER** What test-set log likelihood performance would you get if you simply predicted the next word using a uniform probability distribution over the vocabulary? Please report a concrete numerical value, comparable to the scores in Fig 1a above. Does the ML estimator always beat this "dumb" baseline?

Problem 2

In problem 1, we set α manually to a single value.

Here in problem 2, we'll now explore principled ways to select the value of α to optimize performance given our training set.

Tasks for Code Implementation

2(i): Implement the `calc_per_word_log_evidence` method in the starter code `run_select_alpha.py`, using the formula given above.

2(ii): Edit whatever you need in the `__main__` section of that script to make Figure 2a below

Tasks for Report PDF

2a: Figure for Experiment 2: Selecting α for the Posterior Predictive Estimator

In your report PDF, deliver a figure assessing model selection with 3 panels, one for 3 possible training data sizes: $N/128$, $N/16$, and N. For each dataset size, plot the per-word log evidence of the training set (e.g. the value produced by your calc_log_evidence function, divided by the number of words in the training set) as a function of α, for the log-spaced grid of alpha values suggested in the starter code. On the same axes, overlay the "test set" per-word log probability computed by your posterior predictive estimator at each value of α. If the evidence is a good indicator of which α to select, the two curves should have similar trends in terms of peak performance.

Below this figure in your report PDF, answer the following questions:

- **2b: SHORT ANSWER** A common alternative strategy for selecting α is to (1) reserve a small portion of the train set as a "validation set" and then (2) select the value of α (out of a grid of candidate values) that maximizes the (conditional) log likelihood on the validation set after training on the train set. Describe one advantage of the evidence-on-train approach used in Fig 2a. Then, describe one advantage of the conditional-likelihood-on-validation approach.
