CSE Template

Rohan Mukherjee

October 30, 2024

1.2.2.

1. There were only 6 features for the linguistic features, therefore there is very small room for overfitting since the image can be at most 6-dimensional. On the other hand, the BOW features had 4289 features, which gives an enormous amount of room for overfitting.

2. Here is the table:

Model	Dev Accuracy	Dev Precision	Dev Recall	Dev F1-Score
Logistic Regression BOW	0.73	0.69	0.60	0.64
no lower case	0.72	0.68	0.58	0.63
no replacing rare words with unk	0.73	0.69	0.59	0.64
no removing punctuation	0.73	0.69	0.61	0.65
no removing stop words	0.74	0.70	0.64	0.67

Table 1: Development set performance metrics for different models

3. I tried learning rates in e^{-1} to e^{-5}, batch sizes from 2^3 to 2^{12}, epochs from 2 to 1000, and all normalization techniques as 0/1. The best hyperparameters I found were:

   lower case=True, remove punctuation=True, remove stopwords=False, replace rare words wth unks=False

   num epochs=556, batch size=8, learning rate=0.010508660465402792

4. My idea was to say something along the lines of ”The start of the movie was (really strongly negative word), but in the end I (positive word) it!”. To a human the context of seeing ”the start of the movie” would show us that the negative sentiment is restricted only near the beginning, but that the end experience was positive. A model doesn’t know

this. Via similar reasoning, ”The start of the movie was very postiive word but in the end I negative word it!” would be the same. Other templates I made, following this advice, were: ”I loved the positive word of the movie, but the negative word was bad enough to make me not enjoy it.” ”The negative word of the movie was horrendously terrible, but the positive word was so good that I ended up enjoying it.”

1.3.2

1. Notice that, for any constant c, as $e^x e^c = e^{x+c}$, we have that:

   $$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - c}}{\sum_j e^{x_j - c}} = softmax(x - (c, \ldots, c))_i$$

   The maximum float is around 10^38, and $\ln(10^38)$ 87.6. If even one of these x_i are bigger than 88, we will get overflow. By subtracting the maximum value we can guarantee that this doesn't happen, as every power will be ≤ 0.

2. I tried learning rates in the range e^{-1} to e^{-5}, batch sizes from 2^8 to 2^{12}, epochs from 100 to 2000, and all normalization techniques as 0/1. The best hyperparameters I found were:

   lower case=0, remove punctuation=0, remove stopwords=0, replace rare words wth unks=1, and

   num epochs=1366, batch size=474, learning rate=0.062176524022116327

2.1.2

1.

Pushing to the extremes, if we replaced every word with the ¡unk¿ token, then the probability of seeing the test data would be 1, since every word would just be unknown. The only N grams would be N ¡unk¿s. However, this obviously drastically decreases generation quality, because it would just give us ¡unk¿ over and over (or with less replacement, more often) which doesn't contribute any information.

2.

Here is the table:

¡unk¿ Threshold	Train Perplexity	Dev Perplexity
1	575	∞
3	384	282
5	308	233
7	259	196
9	227	175
10	216	167

Table 2: Perplexities for different ¡unk¿ thresholds

here are the examples for threshold 1:

```
Senator of princess . be , STANLEY pieces the thou displeasure tell my
    you , QUEEN meaning tongue it !
day QUEEN sickly advanced am , <eos>
protector ’d : , some : <eos>
shall told how ? been you ; teeth Yet contrived thou will , if the is
    infold of cursed head
e’er way by can <eos>
```

threshold 3:

```
ye you : when <eos>
hideous Caesar winter ; : piteous that model unrest Her is it crown the
     <eos>
’ longer It MENENIUS , will apprehend . what withal face how thou a is
    prove senseless Mine villain ’s
not thy ’d thou part extremity and ; , my : for till you now : : shed a
     .
his day to by , , say mother purpose stand causes ’s him night <unk> of
     , : Darest free
```

threshold 5:

```
look it <eos>
<eos>
<unk> his From But <unk> KING thee say do ! <unk> and best : , , we
    mock then their
, for their <eos>
would ; They may , our For I request But <eos>
```

threshold 7:

```
, sir : help chamber hath have tell but not King have feeling king <eos
    >
, , : , ’t , ? lesser : less <unk> me That had lives out commission <
    unk> <eos>
her daughter Mantua ; and true the hanging death it her thee , speech
    with thank ? shame have :
king , ? Didst revenged say do What <unk> you love her fault shadow ,
    them , he ! right
the hear . fire my , , And , lord And , , CAPULET and and hearing LADY
    GREGORY noble
```

threshold 9:

```
people , down not ; shame , of : would , me hear They man country
    thereof the <unk> late
than grief : committed , something I WARWICK , not IV for my To !
    MENENIUS well consul , <unk>
kindred least ! comes <unk> <eos>
What is well , Mayor Servant ’t I <unk> These up <eos>
wit honour Richard Which ’ a maid fair , : We been BENVOLIO . can him
    suspicion again both would
```

threshold 10:

```
our <unk> , souls along ISABELLA Shepherd in some Lord lady <unk> be .
    close help <unk> your ’ is
, your this noble this night in not lords ’ll . : out why little <unk> .
     : you well
and ; Do you them mind and here within Edward . Lest that <unk> !
    castle that to MENENIUS <unk>
, . . . . my <eos>
you wife would your <eos>
```

As the threshold is increased, the number of ¡unk¿s goes down, and the perplexity goes down. However, the quality of the generated text gets worse, because there are simply too many words to pick from. It is likely also prone to just regurgitating the training data.

3.

Taking the argmax would be very bad. Instead of sampling from a probability distribution, generating probably new sentences, taking the argmax would just take a carbon copy from the training data. For example, if we were using a 4-gram model and fed it ¡sos¿ ¡sos¿ ¡sos¿ as the prefix, it would always give us the same sentence.

2.2.2

There is a typo in the homework–the total number of terms in $w_{k-N-1}, \ldots, w_{k-1}$ is $k-1-(k-N-1)+1 = k-k-1+1+N+1 = N+1$, while it should be $N-1$. The Markov assumption for an N-gram language model is that the prediction of the next word depends only on the $N-1$ previous words, and all context before that doesn't matter. That is to say,

$$\mathbb{P}(w_k|w_1, \ldots, w_{k-1}) = \mathbb{P}(w_k|w_{k-N+1}, \ldots, w_{k-1})$$

Then by the chain rule,

$$\mathbb{P}(w_1, \ldots, w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k|w_1, \ldots, w_{k-1}) = \prod_{k=1}^{n} \mathbb{P}(w_k|w_{k-N+1}, \ldots, w_{k-1})$$

Where we have handled negative indices by adding ¡sos¿ tokens.

2.3.2

1. For unigram, $\mathbb{P}(w_1, \ldots, w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k) = \prod_{k=1}^{n} \frac{C(w_k)+1}{V}$ For bigram,

   $$\mathbb{P}(w_1 \ldots w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k|w_{k-1}) = \prod_{k=1}^{n} \frac{C(w_{k-1}w_k)+1}{C(w_{k-1})+V}$$

   For trigram,

   $$\mathbb{P}(w_1 \ldots w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k|w_{k-2}, w_{k-1}) = \prod_{k=1}^{n} \frac{C(w_{k-2}w_{k-1}w_k)+1}{C(w_{k-2}w_{k-1})+V}$$

   For 4-gram,

   $$\mathbb{P}(w_1 \ldots w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k|w_{k-3}, w_{k-2}, w_{k-1}) = \prod_{k=1}^{n} \frac{C(w_{k-3}w_{k-2}w_{k-1}w_k)+1}{C(w_{k-3}w_{k-2}w_{k-1})+V}$$

and finally for 5-gram,

$$\mathbb{P}(w_1 \dots w_n) = \prod_{k=1}^{n} \mathbb{P}(w_k | w_{k-4}, w_{k-3}, w_{k-2}, w_{k-1}) = \prod_{k=1}^{n} \frac{C(w_{k-4}w_{k-3}w_{k-2}w_{k-1}w_k) + 1}{C(w_{k-4}w_{k-3}w_{k-2}w_{k-1}) + V}$$

2. Here is the plot of the perplexities:

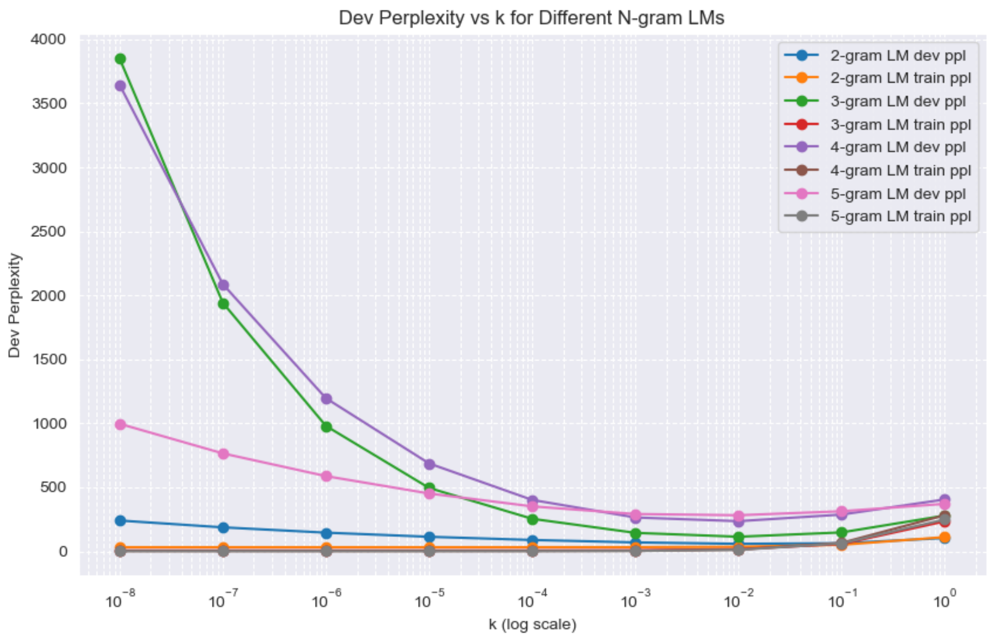
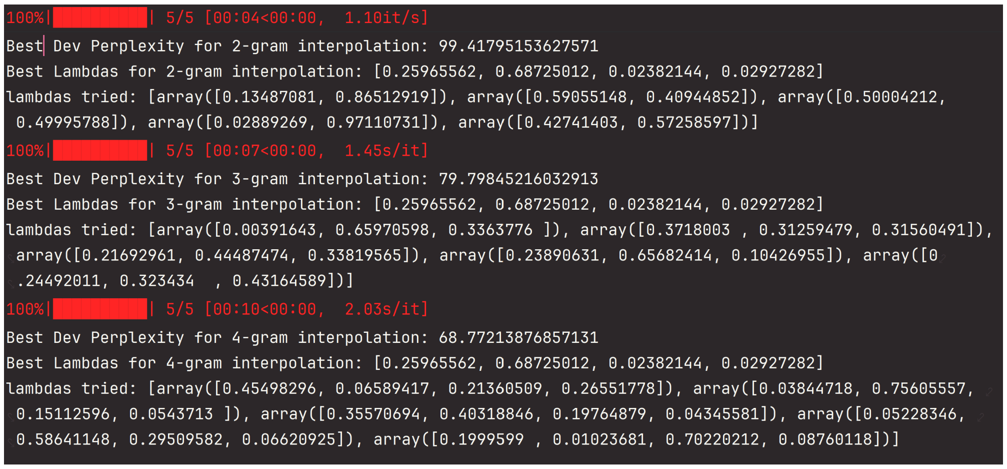


Figure 1: Perplexities of different n-gram models

The best model is seen to be the 2-gram model with learning rate 0.01.

3. For interpolation, I get:



```
100%|██████████| 5/5 [00:04<00:00,  1.10it/s]
Best Dev Perplexity for 2-gram interpolation: 99.41795153627571
Best Lambdas for 2-gram interpolation: [0.25965562, 0.68725012, 0.02382144, 0.02927282]
lambdas tried: [array([0.13487081, 0.86512919]), array([0.59055148, 0.40944852]), array([0.50004212,
 0.49995788]), array([0.02889269, 0.97110731]), array([0.42741403, 0.57258597])]
100%|██████████| 5/5 [00:07<00:00,  1.45s/it]
Best Dev Perplexity for 3-gram interpolation: 79.79845216032913
Best Lambdas for 3-gram interpolation: [0.25965562, 0.68725012, 0.02382144, 0.02927282]
lambdas tried: [array([0.00391643, 0.65970598, 0.3363776 ]), array([0.3718003 , 0.31259479, 0.31560491]),
 array([0.21692961, 0.44487474, 0.33819565]), array([0.23890631, 0.65682414, 0.10426955]), array([0
 .24492011, 0.323434  , 0.43164589])]
100%|██████████| 5/5 [00:10<00:00,  2.03s/it]
Best Dev Perplexity for 4-gram interpolation: 68.77213876857131
Best Lambdas for 4-gram interpolation: [0.25965562, 0.68725012, 0.02382144, 0.02927282]
lambdas tried: [array([0.45498296, 0.06589417, 0.21360509, 0.26551778]), array([0.03844718, 0.75605557,
 0.15112596, 0.0543713 ]), array([0.35570694, 0.40318846, 0.19764879, 0.04345581]), array([0.05228346,
 0.58641148, 0.29509582, 0.06620925]), array([0.1999599 , 0.01023681, 0.70220212, 0.08760118])]
```

Figure 2: Perplexities of different n-gram models with interpolation

I sampled the λs from the Direchlet distribution, which is a uniform distribution for the λs that sum to 1.