A1_17

Evaluation

Part 1: top 5 bigrams with or without smoothing

Without Sentence starting token

```
Top 5 bigrams without smoothing:
   ('href', 'http'): 1.000
   ('tychelle', 'to'): 1.000
   ('hang', 'out'): 1.000
   ('nonexistent', 'social'): 1.000
   ('alex', 'and'): 1.000
```

```
Top 5 bigrams with laplace smoothing:
   ('i', 'feel'): 0.110
   ('feel', 'like'): 0.035
   ('i', 'am'): 0.032
   ('that', 'i'): 0.027
   ('and', 'i'): 0.023
```

```
Top 5 bigrams with kneser-ney smoothing:
   ('don', 't'): 0.970
   ('href', 'http'): 0.970
   ('didn', 't'): 0.958
   ('sort', 'of'): 0.957
   ('supposed', 'to'): 0.918
```

With Sentence starting token as `@`

```
Top 5 bigrams without smoothing:
   ('href', 'http'): 1.000
   ('tychelle', 'to'): 1.000
   ('hang', 'out'): 1.000
   ('nonexistent', 'social'): 1.000
   ('alex', 'and'): 1.000
```

```
Top 5 bigrams with laplace smoothing:
   ('@', 'i'): 0.269
   ('i', 'feel'): 0.110
   ('feel', 'like'): 0.035
   ('i', 'am'): 0.032
   ('@', 'im'): 0.027
```

```
Top 5 bigrams with kneser-ney smoothing:
   ('don', 't'): 0.970
   ('href', 'http'): 0.970
   ('didn', 't'): 0.958
   ('sort', 'of'): 0.957
   ('supposed', 'to'): 0.918
```

Part 2: Reasoning for emotion component

For our bigram final probability we have two components:

- Probability score based on the total bigram occurrences
- Emotion scores generated using the `emotion_score` API provided.

We are taking weighted mean of these components i.e.

```
count_prob = count(w_i, w_i_minus_1) / count(w_i_minus_1)
emotion_score = emotion_scores(wi, w_i_minus_1)[emotion][score]
```

```
final_probability = w1*count_prob + w2*emotion_score {where w1 +
```

> The weights which are `w1` and `w2` depends upon the `emotion_score`. We have set up the thresholds for the `emotion_score` which indicate that "use these weights given this score". The basic intuition is that if the `emotion_score` for a bigram is not very high then reduce the emotion component and increase the count component (which loosely translates to if this bigram have more probability of not being part of the given emotion than being part of it then does not rely too much on this probability and consider the occurrences of this bigram). If bigram is a part of this emotion with high probability then increase the emotion component and reduce the occurrence component.

$$\beta = \frac{w2}{w1}\{where : w1 + w2 = 1\}$$

The thresholds are as follows:

```
if emotion_score > 0.9 then w1 = 0.1 and w2 = 0.9
if emotion_score < 0.5 then w1 = 0.9 and w2 = 0.1
if emotion_score < 0.7 then w1 = 0.6 and w2 = 0.4
else w1 = 0.3 and w2 = 0.7
```

Part 3

2 generated samples for each emotion, for a total of 12 generated samples

Sadness

- i forgot my despair in vain today i find my disappointment that side of dumb when non make is ungrateful hellip and suffer the ugly
- ill do an awful feeling pain of losing at am devastated when they missed yoga because that unwell it dirty src rte emoticons smile tha

Joy

- i did feel and heals them integrated with loreal max factor and delighted in retrospect if ahead of peacefulness her nicely
- id enjoy the beauty plus the support the jasmine green tea next year are likely to educating writers go of pleasantness is steady hands careess

Surprise

- i can never done a surprised us media policy and curiosity is amazing tools met again last five or stunned by being impressed that shocked
- im shocked my funny cuz its only amazing when were people harrass me shocked over so funny to teach me shocked looking it would sing

Fear

- i caught in another intrusion to use headphones sick but seriously feel frightened or uncomfortable etc pp
- i accidentally feel nervous and shaking for fear because arun has such and sensitive nature and indecisive because thats when reading beware here be proved

Anger

- no description feel rebellious i ran away i possibly can insist and plastic slippers for months you who am again though the judge passing sentence
- is killing my hair a lot lately and blowing my time but ill desperately trying something so tortured enough away for hating myself not curl

Love

- i loved for sharing your beloved nakahara mai would like caring in too cheap so delicate pressure on loved ones name for accepting and compassionate

- i feel caring in brave men will contain some sort of hot blanket than fond toward though i wish than fond of caring about supporting

Part 4: Accuracy and macro F1 scores

Accuracy

```
0.8666666666666667
0.8966666666666666
0.8666666666666667
0.8333333333333334
0.8766666666666667
0.8866666666666667
0.9066666666666666
0.8566666666666667
0.8933333333333333
0.86
0.8733333333333333
0.8866666666666667
0.8766666666666667
0.89
0.86
0.87
0.9066666666666666
0.87
0.93
0.88
```

These are the accuracies of our model for 20 different iterations. Each iteration generate 50 sentences for each emotion. Average accuracy for our test is as follows: `0.8793333333333331` .

Classification Report

	precision	recall	f1-score	support
anger	0.94	0.64	0.76	50

fear	0.88	0.92	0.90	50
joy	0.89	0.78	0.83	50
love	0.83	1.00	0.91	50
sadness	0.83	0.96	0.89	50
surprise	0.94	0.98	0.96	50
accuracy			0.88	300
macro avg	0.89	0.88	0.88	300
weighted avg	0.89	0.88	0.88	300

Part 5: Reasoning behind these sentence generation

Define Models for Each Emotion

```
models = {
    "sadness": Bigram_LM(),
    "joy": Bigram_LM(),
    "surprise": Bigram_LM(),
    "fear": Bigram_LM(),
    "anger": Bigram_LM(),
    "love": Bigram_LM()
}
```

We have generated 6 models for each emotion which has initially been initialized with the Bigram_LM class. For each bigram we have called the emotion_scores() corresponding to each bigram and depending on the beta calculation we have populated the bigram probability.

> Refer to Part-2 to better understand how the models are trained.

Choosing the First Token

We have stored the first token in the form of `("@", <first-token>)` for each corresponding sentence.

```
starting_word = random.choices(starting_bigrams, starting_bigram
```

Starting word is determined probabilistically with the above formula.

Sentence generation process

Depending on the first token which was appended in the sentence, the next word is determined depending on the last word appended to the sentence.

```
next_word = sample_next_word(model, sentence[-1])
```

A list of candidates word is generated for each last word in the sentence and then the process is continued using the formula given below.

```
next_word_probs = [model.bigram_probs[(previous_token, word)] fo
return random.choices(candidate_words_list, next_word_probs)[0]
```

We have set the `Max_length = 25` and `Min_length = 10` so until a token with no bigram is found the sentence will continue to form if the length is less than Max_length. If we encounter the last token when sentence is less 10 words then we randomly choose a bigram using the random.choices() function.

Part 6: contributions

> Note: Any part is not actually done by a particular student still we are able to enlist contributions.

Lakshay Chauhan (2021060):

- Task1: BPE_Tokenizer.learn()
- Task2: Bigram_LM.learn()
- Task2: Kneser-Ney.bigram_probability()
- Task2: Bigram emotion scores generation
- Task2: beta optimization

- Task1: debugging
- Task2: debugging

Krishna Somani (2021058)

- Task2: Laplace.bigram_probability()
- Task2: generate_samples()
- Task2: beta optimization
- Task2: min max length optimization,
- Task1: debugging
- Task2: debugging

Mayank Gupta (2021058)

- Task1: BPE_Tokenizer.learn()
- Task2: Unigram_optimization()
- Task2: Bigram_LM.sample_next_word()
- Task2: beta optimisation
- Task2: Entrinsic evaluation
- Task1: debugging
- Task2: debugging

Arnav Singh (2021019)

- Task2: Entrinsic evaluation
- Task1: debugging
- Task2: debugging
- Evaluation_part1: top 5 for each smoothing,
- Task2: unigram emotion scores generation