67658 Natural Language Processing

Exercise 1

Asaf Shul 207042714

Daniel Azulay 311119895

initialize notebook:

```
import spacy
import textacy
import pickle
import numpy as np

from collections import Counter
from datasets import load_dataset

# uncomment to download the module:
# !python -m spacy download en_core_web_sm

# load the data:

nlp = spacy.load("en_core_web_sm")
dataset = load_dataset('wikitext', 'wikitext-2-raw-v1', split='train')
corpus = dataset['text']
```

```
Found cached dataset wikitext
(/Users/asafshul/.cache/huggingface/datasets/wikitext/wikitext-2-raw-
v1/1.0.0/
a241db52902eaf2c6aa732210bead40c090019a499ceb13bcbfa3f8ab646a126)
```

functions:

```
def clean_text(text):
    return 'START ' + ' '.join([c.lemma_ for c in nlp(text) if
c.is_alpha])

def train_language_model(corpus, level=2):
    ngrams = []
    print('counting words...')

    # extract_ngrams:
    for text in corpus:

ngrams.extend(list(textacy.extract.ngrams(nlp(clean_text(text)),
level, filter_stops=False)))

    print('calculating freqs...')
    if level==1:
```

```
        return get_unigram(ngrams)
    if level==2:
        return get_bigram(ngrams)

def get_unigram(ngrams):
    word_counts = dict(Counter([str(w) for w in ngrams]))
    _N = sum(word_counts.values())

    for key in word_counts:
        word_counts[key] = np.log(word_counts[key] / _N)

    return word_counts

def get_bigram(ngrams):
    base_dict = dict(Counter([(str(w[0]), str(w[1])) for w in
ngrams]))
    freq_dict = {}

    # format to bigram dict:
    for key, count in base_dict.items():
        base_word, next_word = key
        if base_word not in freq_dict:
            freq_dict[base_word] = {next_word : count}
        elif next_word not in freq_dict[base_word]:
            freq_dict[base_word][next_word] = count
        else:
            freq_dict[base_word][next_word] += count

    # change count to relative probability:
    for word_counts in freq_dict.values():
        _N = sum(word_counts.values())
        for key in word_counts:
            word_counts[key] = np.log(word_counts[key] / _N)

    # return the model:
    return freq_dict
```

1.

Train maximum-likelihood unigram and bigram language models based on the above training data.

```
%%time

# unigram:
uni_model = train_language_model(corpus, 1)
```

```
counting words...
calculating freqs...
CPU times: user 12min 2s, sys: 3.13 s, total: 12min 6s
Wall time: 12min 12s
```

```
%%time

# bigram:
bi_model = train_language_model(corpus, 2)
```

```
counting words...
calculating freqs...
CPU times: user 11min 48s, sys: 4.2 s, total: 11min 53s
Wall time: 11min 55s
```

```
# save models to pickle file:

# with open('unigram.pickle', 'wb') as file:
#     pickle.dump(uni_model, file, protocol=pickle.HIGHEST_PROTOCOL)

# with open('bigram.pickle', 'wb') as file:
#     pickle.dump(bi_model, file, protocol=pickle.HIGHEST_PROTOCOL)

# load models:

# with open('unigram.pickle', 'rb') as fp:
#     uni_model = pickle.load(fp)

# with open('bigram.pickle', 'rb') as fp:
#     bi_model = pickle.load(fp)
```

1. Using the bigram model, continue the following sentence with the most probable word predicted by the model: " I have a house in ...".

```
def pred_bigram(model, sentance):
    last_word = sentance.split(' ')[-1]
    probs = bi_model[last_word]
    idx = np.argmax(list(probs.values()))
    return list(probs.keys())[idx]
```

```
sentance = 'I have a house in'
pred_bigram(bi_model, sentance)
```

```
'the'
```

1. Using the bigram model:

(a) compute the probability of the following two sentences (for each sentence separately).

(b) compute the perplexity of both the following two sentences (treating them as a single test set with 2 sentences).

- Brad Pitt was born in Oklahoma
- The actor was born in USA

```
def calc_sentance_log_prob_bigram(sentance, bi_model):
    sentance_arr = clean_text(sentance).split(' ')
    prob = 0
```

```
    for word, next_word in zip(sentance_arr[:-1], sentance_arr[1:]):
        if word in bi_model and next_word in bi_model[word]:
            prob += bi_model[word][next_word]
        else:
            return -np.inf

    return round(prob, 3)
```

```
sentance1 = 'Brad Pitt was born in Oklahoma'
sentance2 = 'The actor was born in USA'

prob1 = calc_sentance_log_prob_bigram(sentance1, bi_model)
prob2 = calc_sentance_log_prob_bigram(sentance2, bi_model)

print(f'- prob for "{sentance1}" is: {prob1}')
print(f'- prob for "{sentance2}" is: {prob2}')
```

```
- prob for "Brad Pitt was born in Oklahoma" is: -inf
- prob for "The actor was born in USA" is: -29.687
```

```
print(f'prob for both "{sentance1}" and "{sentance2}" is: {prob1 +
prob2}')
```

```
prob for both "Brad Pitt was born in Oklahoma" and "The actor was born
in USA" is: -inf
```

1. Now we use linear interpolation smoothing between the bigram model and unigram model with:

- λbigram = 2/3
- λ_unigram = 1/3

using the same training data.

Given this new model, compute the probability and the perplexity of the same sentences such as in the previous question.

```
def calc_sentance_log_prob_interpulation(sentance, uni_model,
bi_model, lam_bi, lam_uni):
    sentance_arr = clean_text(sentance).split(' ')
    prob = 1

    for word, next_word in zip(sentance_arr[:-1], sentance_arr[1:]):
        if (word in bi_model and next_word in bi_model[word]) and
(next_word in uni_model):
            prob *= (lam_bi * np.exp(bi_model[word][next_word])) +
(lam_uni * np.exp(uni_model[next_word]))

        elif word in bi_model and next_word in bi_model[word]:
            prob *= (lam_bi * np.exp(bi_model[word][next_word]))
```

```
        elif next_word in uni_model:
            prob *= (lam_uni * np.exp(uni_model[next_word]))

        else:
            return -np.inf

    return np.log(prob)

lam_bi = 2/3
lam_uni = 1/3

# probability:
sents1_prob = calc_sentance_log_prob_interpulation(sentance1,
uni_model, bi_model, lam_bi, lam_uni)
sents2_prob = calc_sentance_log_prob_interpulation(sentance2,
uni_model, bi_model, lam_bi, lam_uni)
print(f'- prob for "{sentance1}" is: {sents1_prob}')
print(f'- prob for "{sentance2}" is: {sents2_prob}')
```

```
- prob for "Brad Pitt was born in Oklahoma" is: -36.20826629122097
- prob for "The actor was born in USA" is: -31.017958929308932
```

```
# perplexity:
M = len(clean_text(sentance1).split(' ') +
clean_text(sentance2).split(' ')) - 2
print(f'perplexity in test set is is: {np.exp(-((sents1_prob +
sents2_prob) / M))}')
```

```
perplexity in test set is is: 271.0180530398554
```