Winter 2024 Information Retrieval HW2

Hao Jun Chen

February 21, 2024

1 Maximum likelihood estimation for statistical language models with proper smoothing

1. Linear Interpolation Smoothing

```
    unigram_percentage: a dictionary containing the percentage of each word
    bigram_freq_dict: a dictionary contain the percentage of all possible
        bigram
    query_word: w_i
    given_word: w_{i-1}
    tokens_count: a dictionary contain count of each token
```

Listing 1: parameters

```
    def LTS(unigram_percentage, bigram_freq_dict,query_word, given_word,
        tokens_count):
        bigram = (given_word,query_word)
        #c(w_i-1, w_i)
        if bigram in bigram_freq_dict:
            bigram_freq = bigram_freq_dict[bigram]
        else:
            bigram_freq = 0
        #c(w_i)
        given_word_freq = tokens_count[given_word]
        #(1-lamda)* c(w_i-1*w_i)/c(w_i-1)
        mle = (bigram_freq/given_word_freq) * 0.1
        #lamda*p(w_i)
        parameter = 0.9 * unigram_percentage[query_word]
        return mle + parameter
```

Listing 2: Implementation of Linear Interpolation Smoothing

2. Absolute Discounting Smoothing

```
    unigram_percentage: a dictionary containing the percentage of each word
    bigram_freq_dict: a dictionary containing the percentage of all possible
        bigram
    count_given_word_given_query_word: a dictionary containing the count of
        unique w_{i-1} giving a w_i
    count_big_freq_of_given_word: a dictionary containing the count of bigram
        that start with w_{i-1}
    query_word: w_i
    given_word: w_{i-1}
```

Listing 3: parameters

```
    def ADS(unigram_percentage, bigram_freq_dict,
        count_given_word_given_query_word, count_big_freq_of_given_word,
        query_word, given_word):
        bigram = (given_word,query_word)
        if bigram in bigram_freq_dict:
            #max(c(w_i,w_i-1),0)
            bigram_freq = max(bigram_freq_dict[bigram] - 0.1,0)
        else:
            bigram_freq = 0
        if query_word in count_given_word_given_query_word:
            #number of w_i-1 given the w_i:
            unique_count = count_given_word_given_query_word[query_word] * 0.1
        else:
            unique_count = 0
        #unigram prob of given_word
        given_word_freq = unigram_percentage[given_word]
        #number of bigram starting with w_i-1
        given_word_count = count_big_freq_of_given_word[given_word]
        return (bigram_freq + (unique_count * given_word_freq)) /
            given_word_count
```

Listing 4: Implementation of Linear Interpolation Smoothing

3. top 10 words that start with goods using Linear Interpolation Smoothing

Percentage	Word
0.0526756205746076	the
0.03541420686717264	and
0.031395063767803884	i
0.023807860627484093	a
0.02011460692266228	to
0.018442686910707815	but
0.01785910013690753	it
0.017255957345376327	wa
0.014689357581130841	of
0.011829494926671551	for

4. top 10 words that start with goods using Absolute Discounting Smoothing

Percentage	Word
0.10000110228499007	but
0.06652223665640758	and
0.05611250382583506	i
0.05459198583043281	the
0.03795093623526897	as
0.02961551778895204	food
0.019583100540012974	it
0.018205159410044695	thing
0.017995565982029685	for
0.017130254625017254	too

5. Observation about the top words We can see that the overall probability of Linear Interpolation Smoothing is less than Absolute Discounting Smoothing. This makes sense since Absolute Discounting Smoothing adds more weight to those unseen words.

2 Generate text documents from a language model

1. documents generation by Linear Interpolation Smoothing

```
    def LTS_next_word(unigram_percentage, bigram_freq_dict, given_word,
        tokens_count):
        words = []
        prob = []
        for token in tokens_count:
            query_word = token
            LTS_value = LTS(unigram_percentage, bigram_freq_dict, token,
                given_word, tokens_count)
            words.append(query_word)
            prob.append(LTS_value)
        prob = np.array(prob)
        prob /= prob.sum()
        random_index = np.random.choice(len(words), p=prob)
        return (words[random_index], prob[random_index])
```

Listing 5: Linear Interpolation Smoothing Sample next word

```
    def LTS_genrate_doc(unigram_percentage, bigram_freq_dict, tokens_count,
        doc_len, total_doc):
        LTS_docs = []
        uni_words = list(unigram_percentage.keys())
        uni_prob = list(unigram_percentage.values())
        uni_prob = np.array(uni_prob)
        uni_prob /= uni_prob.sum()
        for i in range(total_doc):
            start_word = uni_words[np.random.choice(len(uni_words), p=uni_prob
                )]
            doc = [start_word]
            likelihood = 0
            for i in range(1, doc_len):
                given_word = doc[i-1]
                cur_word = LTS_next_word(unigram_percentage, bigram_freq_dict,
                    given_word, tokens_count)
                doc.append(cur_word[0])
                if likelihood == 0:
                    likelihood = cur_word[1]
                else:
                    likelihood = likelihood * cur_word[1]
            doc_item = (doc, likelihood)
            LTS_docs.append(doc_item)
        return LTS_docs
```

Listing 6: Linear Interpolation Smoothing Text Generation

2. documents generation by Absolute Discounting Smoothing

```
    def ADS_next_word(unigram_percentage, bigram_freq_dict,
        count_given_word_given_query_word, count_big_freq_of_given_word,
        tokens_count, given_word):
        words = []
        prob = []
        for token in tokens_count:
            query_word = token
            ADS_value = ADS(unigram_percentage, bigram_freq_dict,
                count_given_word_given_query_word,
                count_big_freq_of_given_word, token, given_word)
```

```
            words.append(query_word)
            prob.append(ADS_value)
        prob = np.array(prob)
        prob /= prob.sum()
        random_index = np.random.choice(len(words), p=prob)
        return (words[random_index], prob[random_index])
```

Listing 7: Absolute Discounting Smoothing Sample next word

```
    def ADS_generate_doc(unigram_percentage, bigram_freq_dict,
        count_given_word_given_query_word, count_big_freq_of_given_word,
        tokens_count, doc_len, total_doc):
        ADS_docs = []
        uni_words = list(unigram_percentage.keys())
        uni_prob = list(unigram_percentage.values())
        uni_prob = np.array(uni_prob)
        uni_prob /= uni_prob.sum()
        for i in range(total_doc):
            start_word = uni_words[np.random.choice(len(uni_words), p=uni_prob
                )]
            doc = [start_word]
            likelihood = 0
            for i in range(1, doc_len):
                given_word = doc[i-1]
                cur_word = ADS_next_word(unigram_percentage, bigram_freq_dict,
                     count_given_word_given_query_word,
                    count_big_freq_of_given_word, tokens_count, given_word)
                doc.append(cur_word[0])
                if likelihood == 0:
                    likelihood = cur_word[1]
                else:
                    likelihood = likelihood * cur_word[1]
            doc_item = (doc, likelihood)
            ADS_docs.append(doc_item)
        return ADS_docs
```

Listing 8: Absolute Discounting Smoothing Text Generation

3. Documents generate by Linear Interpolation Smoothing

- docs: well come bunch rogu is tasti chef lettuc keep tri someth and after discuss definit some dish spot to yellow, doc's len: 20 likelihood: 7.124003206094247e-61
- docs: about meltinyourmouth i will lo 11am or a hi qualiti we menu total chicagostyl chocol us up our ever order, doc's len: 20 likelihood: 6.232641838443836e-59
- docs: wa here you great thi word less the tough sure choic just fed better breakfast porcini make can NUM and, doc's len: 20 likelihood: 6.727898936650642e-57
- docs: cider it it again it line there hot establish with usual reduct and here ferdi one fear the dinner bathroom, doc's len: 20 likelihood: 1.49088211419103e-55
- docs: NUM line also i attempt graini in servic on of that white complaint other an my i i love all, doc's len: 20 likelihood: 3.0983751711893755e-51
- docs: the backyard light which the breakfast of an that out the much and decent the NUM fri standard for nearbi, doc's len: 20 likelihood: 4.422054196254075e-50
- docs: wo thi did nt up a they will could chees bean and so thi is the been salami ago of, doc's len: 20 likelihood: 6.171903113372097e-47
- docs: get peoplewatch cheap were tast fresh real for at a trio need thank wa i say from char pizza were, doc's len: 20 likelihood: 3.290881691023122e-56

- docs: be each re if thi wa as i littl just plancha littl parti afford anyth stilton the the dmk sex, doc's len: 20 likelihood: 6.2711725964174764e-58
- docs: wa feel wa huge are by in done hey or a in s chicken or and to you to those, doc's len: 20 likelihood: 7.500207471690346e-48

4. Documents generate by Absolute Discounting Smoothing

- docs: befor thi place to read review for lime and tri thi wa pack and tens ontheedg lunch servic wa here, doc's len: 20 likelihood: 5.0061798164191414e-36
- docs: chees peppercini s piggin n outlik in gener portion and by more the hopleaf great thing and i ve tri, doc's len: 20 likelihood: 7.853988237750522e-40
- docs: chose the sweet milk and like a la ye you wait i ca nt find a great night i saw, doc's len: 20 likelihood: 2.2671021724940552e-36
- docs: adam came here my favorit restaur is by for all the origin benedict good but other tabl and eat if, doc's len: 20 likelihood: 9.331346971068856e-37
- docs: wa nt a must nice in your like s3media4akyelpcdncom i had never had a go we arriv either so excit, doc's len: 20 likelihood: 1.4036746590671078e-44
- docs: frite the stroller that she took marilyn to the 20cours menu is pretti damn good not mean that delici i, doc's len: 20 likelihood: 1.7052899138173243e-41
- docs: it is a tabl bread pud but nice wood wa were ahead you upstair but then go with friend and, doc's len: 20 likelihood: 4.634161316343349e-44
- docs: with porki ve been pleas again and waffl yum mayo the servic is soso chicken wing titan blt burger is, doc's len: 20 likelihood: 3.126965672200815e-42
- docs: last matzo ball of the mushroom scrambl but would finish first took me wrong food and refresh get thing to, doc's len: 20 likelihood: 2.85610507294161e-44
- docs: insteadit s good and the big point for a go too thi hipster crowd expens either after everyth better than, doc's len: 20 likelihood: 3.388714067460693e-42

3 Reading Assignment

As the paper said, "Search engines may wish to favor accuracy over satisfaction given the potentially serious ramifications of an incorrect answer", I think this can be an approach for search engines. When it comes to essential information like health-related topics, it's more important to display true information than the information the user wants to hear. In terms of healthcare information, one method I can think of is to raise the weight of documents from high-credit sources from websites or organizations with high reputations. Therefore, the rank of the document can be higher. On the opposite, search engines can also rank those low-credit sources in a lower position even though the document matches the query better. In some extreme cases, marking the result as an untrustworthy source in the search results page prevents users from entering it. This might help those unsure users to find correct answers as the paper found in the study, "If people are unsure (i.e., have equal belief in yes and no) they are almost twice as likely to move toward a positive answer rather than negative following searching." If a wrong source with a positive answer to a user's question ranks high, this might harm the user even satisfies the information need.

4 Extra Credits

LLM vs LM: In terms of implementation, LLM requires many more parameters than traditional LM. For example, in a unigram, we only need the word frequency, size of vocabulary, and probability of each word. In Bigram, we have more complex parameters such as different smoothing methods, bigram frequency and probability, starting token, etc. As for LLM, it's far more complex. For instance, GPT-3 has 175 billion parameters. More than that, GPT-3 is also far more complex in terms of architecture. While n-grams are

just a signal function, GPT is a neural network model that contains many layers. Another important feature of GPT-3 is context understanding. GPT is trained with source that with context with logic while n-grams only able to train n words. This is a significant difference in their function.

References:

https://chat.openai.com/

https://www.techtarget.com/whatis/definition/large-language-model-LLM

https://en.wikipedia.org/wiki/Generative_pre-trained_transformer#/media/File:Full_GPT_architecture.png