CS7.401: Introduction to NLP | Assignment 1

Author: Aditya Kumar Singh
ID: 2021701010

February 7, 2022

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Contents

Question 1 - Tokenization

You have been given a twitter corpus for cleaning. Your task is to design a tokenizer using `regex`, which you will later use for smoothing as well.

1. Create a Tokenizer to handle following cases:
   (a) Word Tokenizer
   (b) Punctuation
   (c) URLs
   (d) Hashtags (#manchesterisred)
   (e) Mentions (@john)
2. For the following cases, replace the tokens with appropriate placeholders:
   (a) URLs: <URL>
   (b) Hashtags: <HASHTAG>
   (c) Mentions: <MENTION>

Apart from these, you are also encouraged to try other tokenization and placeholder substitution schemes based on your observations from the corpora used for the smoothing task to achieve a better language model. You'll find percentages, age values, expressions indicating time, time periods occurring in the data. You're free to explore and add multiple such reasonable tokenization schemes in addition from the ones listed above. Specify any such schemes you use in the final README.

Observations:

Twitter corpus (or `general-tweets.txt`) consists of 2000 tweets of which some of them consists of unrecognisable characters (maybe they are the unicodes of EMOJIS') and some words are written improperly (e.g., *bois* instead of *boys*, *ttyl* for *talk to you later*). In general this corpus is *heavily* dirty and need some serious cleaning. Plus, due to this corpus only, we have come across a lot of cases which a human can understand but not the machines. For which some we devised some preprocessing methods detailed in following **approach & code section**.

Approach & Code:

All the functions below use `RegEx` expressions to do cleaning of the sentence (or `txt` according to the argument in each function). Import `RegEx` in `python3` using `import re`. If not installed, install it by typing `pip3 install regex` in your terminal (in your desired environment if there is any).

1. Replace Hashtags #

```
def replaceHashtags(self, txt):
    return re.sub('\#[a-zA-Z]\w+', '<HASHTAG>', txt)
```

2. Replace Emails

```
def replaceEMAIL(self, txt):
    return re.sub(r'\S*[\w\~\-]\@[\w\~\-]\S*', r'<EMAIL>', txt)
```

3. Replace URL

```
def replaceURL(self, txt):
    return re.sub(r'(https?:\/\/|www\.)?\S+[a-zA-Z0-9]{2,}\.[a-zA-Z0-9]{2,}\S+',
    ↪  r'<URL>', txt)
```

4. Replace MENTIONS

```
def replaceMentions(self, txt):
    return re.sub('@\w+', '<MENTION>', txt)
```

5. UPPER to lower

```
def upperToLower(self, txt): return txt.lower()
```

6. Remove Footnotes

```
def removeFootNotes(self, txt): return re.sub('\[.*\s+-\s+.*\]', '<FOOTNOTE>', txt)
```

7. Replace Punctuations

```
def replacePunctuation(self, txt):
    return re.sub(r'(!|"|\#|\$|%|&|\'|\(|\)|\*|\+|,|-|\.|\/|:|;|<|=|>|\?|@|\[|\\|\]|\
    ↪  ^|_|[U+FFFD]|\{|\||\}|~)\1{1,}', r'\1',
    ↪  txt)
```

8. Replace DATE and TIME

```
def replaceDateTime(self, txt):
    txt = re.sub(
        r'\d{2,4}\-\d\d-\d{2,4}|\d{2,4}\/\d\d\/\d{2,4}|\d{2,4}:\d\d:?\d{2,4}',
        ↪  '<DATE>', txt)
    return re.sub(r'\d+:\d\d:?\d{0,2}?( am|am| pm|pm)', r'<TIME>', txt)
```

9. Replace MOBILE Number

```
def replaceMobileNumber(self, txt):
    # https://blog.insycle.com/phone-number-formatting-crm
    return re.sub(r'[\+0-9\-\(\)\.]{3,}[\-\.]?[0-9\-\.]{3,}', r'<MOB>', txt)
```

10. Replace Numericals

```
def replaceNumericals(self, txt):
    return re.sub(r'(?<=\s)[\:\.]?\d*[\:\.]?\d*[\:\.]?(?=\s)', r'<NUM>', txt)
```

11. Replace Alphabets

```
def replaceAlphabets(self, txt): return re.sub(r'(\w)\1{2,}', r'\1\1', txt)
```

12. Replace Apostrophe'

```
def replaceApostrophe(self, txt_line):
    # Replacing n't with not
    t1 = re.sub(r'can\'t', r'can not', txt_line)
    t1 = re.sub(r'won\'t', r'will not', txt_line)
    t1 = re.sub(r'([a-zA-Z]+)n\'t', r'\1 not', txt_line)
    # Replacing That's, he's, she's etc. with <root> is.
    t1 = re.sub(r'([a-zA-Z]+)\'s', r'\1 is', t1)
    # Replacing i'm.
    t1 = re.sub(r'([iI])\'m', r'\1 am', t1)
    # Replacing we've, i've.
    t1 = re.sub(r'([a-zA-Z]+)\'ve', r'\1 have', t1)
    # Replacing i'd, they'd.
    t1 = re.sub(r'([a-zA-Z]+)\'d', r'\1 had', t1)
    # Replacing i'll, they'll.
    t1 = re.sub(r'([a-zA-Z]+)\'ll', r'\1 will', t1)
    # Replacing we're, they're.
    t1 = re.sub(r'([a-zA-Z]+)\'re', r'\1 are', t1)
    # Replacing tryin', doin'.
    t1 = re.sub(r'([a-zA-Z]+)in\'', r'\1ing', t1)
    return t1
```

13. Replace Special Characters embedded in Words

```
def replaceSpecialCharsFromWords(self, txt, flag=False):
    # handle Hyphen words, e.g., short-sightedness to short sightedness
    t1 = re.sub(r'([a-zA-Z]+)-([a-zA-Z]+)', r'\1 \2', txt)
    # Again scan for numericals like 29-year-old.
    t1 = self.replaceNumericals(t1)
    # Put spaces around different punctuations.
    t1 = re.sub(
        r'([\*\-\#\%\!\"\$\&\'\(\)\+\,\-\.\/\:\;\=\?\@\[\\\]\^\_\[U+FFFD]\{\|\}\~])',
        ↪  r' \1 ', t1)
    # Remove punctuations as they don't put any value in N-gram.
    if flag:
        t1 = re.sub(r'(?<=[a-z0-9\s\*\-\#\%\!\"\$\&\'\(\)\+\,\-\.\/\:\;\=\?\@\[\\\]\^
        ↪  \_\[U+FFFD]\{\|\}\~\<\>])[\*\-\#\%\!\"\$\&\'\(\)\+\,\-\.\/\:\;\=\?\@\[\\
        ↪  ]\^\_\[U+FFFD]\{\|\}\~\<\>](?=[a-z0-9\s\*\-\#\%\!\"\$\&\'\(\)\+\,\-\.\/\:
        ↪  \;\=\?\@\[\\\]\^\_\[U+FFFD]\{\|\}\~\<\>])', r'',
        ↪  t1)
    # Also remove spaces in between words or symbols(punctuations) ocurring twice.
    return re.sub(r'\s{2,}', r' ', t1.strip())
```

14. Add sentence starting tag <SOS> and ending tag <EOS>

```
def addSOSEOS(self, txt):
    return "<SOS> " + txt.strip() + " <EOS>\n"
```

After pre-processing is done, save the clean-corpus and the vocabulary for future reference.

Example Snippet:

```
######################## Original ########################
Bumping dj sefs mixtape nowww this is my music new skooooool
#ieroween THE STORY OF IEROWEEN! THE VIDEO ->>>>>>>>>>>>>>>>>>>>>>
↪  http://bit.ly/2VFPAV <<<< JUST FOR FRANK !!! [U+FFFD][U+FFFD]
```

```
trick or treating at the mall today; ZOO! last year we had left-overs, this year we
↪  ran out!
@Ussk81 PMSL!!! I try not to stare but I can't help it, like compulsive viewing!!
@Sc0rpi0n676 btw - is there a remote chance i will see you later?
So... was that my invite to whoop ur ass? Sounded like it. RT: @therealPRYSLEZZ:
↪  @gylliwilli it made nox & I just go out & buy rockband 2.
```

```
######################## PreProcessed ##########################
<SOS> bumping dj sefs mixtape noww this is my music new skool <EOS>
<SOS> <HASHTAG> the story of ieroween ! the video - > <URL> < just for frank ! ã§ <EOS>
<SOS> trick or treating at the mall today ; zoo ! last year we had left overs , this
↪  year we ran out ! <EOS>
<SOS> <MENTION> pmsl ! i try not to stare but i ca not help it , like compulsive
↪  viewing ! <EOS>
<SOS> <MENTION> btw - is there a remote chance i will see you later ? <EOS>
<SOS> so . was that my invite to whoop ur ass ? sounded like it . rt : <MENTION> :
↪  <MENTION> it made nox & i just go out & buy rockband <NUM> <EOS>
```

Question 2 - Smoothing

You have been given two corpus: EuroParl corpus, and Medical Abstracts corpus. Your task is to design Language Models for both of these corpora using smoothing. Ensure that you use the tokenizer created in task 1 for this task.

1. Create language models with the following parameters:
   (a) On EuroParl corpus:
      i. LM 1: tokenization + 4-gram LM + Kneyser-Ney smoothing
      ii. LM 2: tokenization + 4-gram LM + Witten-Bell smoothing
   (b) On Medical Abstracts corpus:
      i. LM 3: tokenization + 4-gram LM + Kneyser-Ney smoothing
      ii. LM 4: tokenization + 4-gram LM + Witten-Bell smoothing
2. For each of these corpora, create a test set by randomly selecting 1000 sentences. This set will not be used for training the LM.
   (a) Calculate perplexity score for each sentence of EuroParl corpus and Medical Abstracts corpus for each of the above models and also get average perplexity score/corpus/LM on the train corpus
   (b) Report the perplexity scores for all the sentences in the test set. Report the perplexity score on the test sentences as well, in the same manner above.
3. Compare and analyse the behaviour of the different LMs and put your analysis and visualisation in a report

---

Observations:

Both EuroParl and Medical corpus consists of relatively clean sentences as compared to Twitter corpus. Both have **same count** of sentences from which *randomly* chosen 1000 sentences will be reserved for testing and the rest will be for training (for constructing $n-$gram frequency table).

Now coming upon algorithms, both use `recursive interpolation` as its core idea. But difference lies on the part of what *weights* they are using while going in recursion.

1. Kneser-Ney uses *absolute discounting* = **d** to account (i.e., reserve some small probability or frequency) for the unseen texts by chopping off some frequency for seen texts. The intuition is that since we have good estimates already for the very high counts, a small discount **d** won't affect them much. It will mainly modify the smaller counts, for which we don't necessarily trust the estimate anyway. Whereas in Witten-Bell smoothing discounting does happen by reserving some probability. For unseen tokens a uniform probability score = $\frac{1}{V}$, where V = vocabulary size, is assigned.

2. Secondly, Kneser-Ney uses the concept of *Continuation count* for lower order n-grams during recursion which basically estimate the number of different contexts (or history) on which word **w** has appeared in, that is, the number of n-gram types it completes. We hypothesize that words that have appeared in more contexts in the past are more likely to appear in some new context as well. While Witten-Bell doesn't consider how many unique history are there with the concerned token. It relies upon the token frequency irrespective of its context, due to which an unordered sentence may seem more likely with this smoothing than Kneser-Ney smoothing (e.g., "The am I King" may seem more probable with Witten-Bell than Kneser-Ney (for unigram), while "I am the King" will show same with Witten-Bell but higher probabilty with Kneser-Ney (or lower perplexity)).

Next, we'll discuss our overall approach and what steps we exceute with given smoothing techniques to obtain the *Perplexity* score.

Approach & Code:

Steps:

1. Clean the given corpus using `tokenizer.py` script which include all the functions discussed above. In return we'll have `list` of sentences.

2. Now split the list obtained above into *training* and *test* set (|test-set| = 1000).

3. Construct frequency table (a `dictionary`) for n-grams, $(n-1)$-grams, ..., upto unigrams that stores the # of occurences from training set. But before that append $n-2$ starting tags (`<SOS>`) at beginning if we're constructing n-grams from it.

4. **Inferencing:** Take each sentence (either from training set or test set) and append required starting tags. Then split it into n-grams, feed each n-gram to language model (Kneser-Ney or Witten-Bell) and obtain the *perplexity score* by taking inverse N-th root of product of all n-grams in that sentence.

Kneser-Ney Recursive Algorithm:

$$P_{KN}\left(w_i|w_{i-n+1:i-1}\right) = \frac{max\left(c_{KN}(w_{i-n+1:i}) - d, 0\right)}{\sum_v c_{KN}(w_{i-n+1:i-1}v)} + \lambda_{(w_{i-n+1:i-1})} P_{KN}\left(w_i|w_{i-n+2:i-1}\right) \tag{1}$$

- Below is an overview of our `kneserNey` algorithm which in itself depends on 4 helper functions which one can find in `language_model.py` script.

- When $\lambda = 0$, then first term in 1 is also 0. Because both have same denominator and λ can only be 0 when it's denominator = 0. Plus, the second term in its numerator that is multiplied with $d = 0.75$ also becomes 0, as the expression are similar upto opertaions (i.e., `counting` and `suumming frequencies`). And since λ have same definition for all recursion steps, then it wont't get 0 at lower order, because chance of occusring a higher order n-grams is less than it lower order *subset*. If a particular lower n-gram doesn't exist then higher too doesn't exist.

```
def kneserNey(history, current, recur_step, cent_dict):
    """
    Parameters
    ----------
    history : Tuple
        Tuple consisting of past words.
    current : String
        The word whose likelihood is to be estimated.

    Returns
    -------
    probability of happening of current word given the past.
    """
```

```
    # n ==> for n-grams
    n = len(history.split())+1
    # Check if current word is in VOCAB
    if current not in cent_dict[1]:
        return len(list(filter(lambda x: cent_dict[n][x] == 1,
        ↪  cent_dict[n])))/len(cent_dict[n])
        # return 0.75/sum(cent_dict[1].values())
    # base condition --- Empty String
    if n == 1:
        return 0.25/len(cent_dict[1]) + 0.75/sum(cent_dict[1].values())
            # 0.75*len(dict(filter(lambda item: item[1]>0,
            # cent_dict[1].items())))/(tot_unigram_counts*len(cent_dict[1]))
            # As V cancels out with the NUMerator.
    # c_KN = count or continuation_count
    if recur_step == 1:
        try:
            first_term = max(Cnt(" ".join([history, current]), cent_dict)-0.75,
            ↪  0)/sum_of_counts(history, cent_dict)
        except ZeroDivisionError:
            # Handle 0/0 form
            first_term = 0
    else:
        try:
            first_term = max(cont_count(current, n, cent_dict) - 0.75, 0)/len(cent_dict[n
])
        except ZeroDivisionError:
            first_term = 0
    # Define Lambda
    try:
        lamb = (0.75/sum_of_counts(history, cent_dict))*count_of_positives(history,
        ↪  cent_dict)
    except ZeroDivisionError:
        # Handle 0/0 error.
        # no point of doing recursion further.
        return len(list(filter(lambda x: cent_dict[n][x] == 1,
        ↪  cent_dict[n])))/len(cent_dict[n])
    # New history for further step...
    new_hist = " ".join(history.split()[1:])
    sec_term = lamb*kneserNey(new_hist, current, recur_step+1, cent_dict)
    # NOw combine all the terms to get final term.
    return first_term + sec_term
```

Witten-Bell Recursive Algorithm:

Mathematical expression:

$$P_{WB}\left(w_i|w_{i-n+1:i-1}\right) = \lambda_{(w_{i-n+1:i-1})}P_{MLE}\left(w_i|w_{i-n+1:i-1}\right) + \left(1-\lambda_{(w_{i-n+1:i-1})}\right)P_{WB}\left(w_i|w_{i-n+2:i-1}\right) \quad (2)$$

Below is an overview of our `wittenBell` algorithm. For more refer `language_model.py` script.

```
def wittenBell(history, current, cent_dict):
    """
    Parameters
    ----------
    history : Tuple
        Tuple consisting of past words.
    current : String
        The word whose likelihood is to be estimated.

    Returns
    -------
    probability of happening of current word given the past.
```

```
    """
    # Define BASE condition for this recursive function.
    n = len(history.split()) + 1
    # base condition --- Empty String
    if n == 1:
        if current in cent_dict[1]:
            return Cnt(current, cent_dict)/sum(cent_dict[1].values())
        return len(list(filter(lambda x: cent_dict[1][x] == 1,
        ↪  cent_dict[1])))/len(cent_dict[1])

    # define lambda parameter
    try:
        lamb = count_of_positives(history, cent_dict)/(count_of_positives(history,
        ↪  cent_dict) + sum_of_counts(history, cent_dict))
    except ZeroDivisionError:
        # again lambda consists of term in denom which is similar to denom of pML
        # Due to which lamb=ZeroDivisionError iff pML=ZeroDivisionError
        return len(list(filter(lambda x: cent_dict[n][x] == 1,
        ↪  cent_dict[n])))/len(cent_dict[n])

    # The first term in WITTEN-Bell expression and it's exception is handled above.
    pML = Cnt(" ".join([history, current]), cent_dict)/sum_of_counts(history, cent_dict)

    # Now arraning all above expressions.
    new_hist = " ".join(history.split()[1:])
    return (1 - lamb)*pML + lamb*wittenBell(new_hist, current, cent_dict)
```

In both the scripts `cent_dict` is the dictionary that stores all n-grams frequency table.

Results & Analysis

Below are the snippets of *perplexity* score from `text` file of different models.

1. **LM1:** Tokenization + 4-gram LM + Kneyser-Ney smoothing (For EuroParl Corpus)

```
        ################## TRAIN PART #################
        Average Perplexity Score: 12.982
        <SOS> resumption of the session <EOS>    PP Score = 10.287
        <SOS> i declare resumed the session of the european parliament adjourned on
        ↪  friday <NUM> december 199 and i would like once again to wish you a happy
        ↪  new year in the hope that you enjoyed a pleasant festive period <EOS>
        ↪  PP Score = 4.982
        <SOS> although as you will have seen the dreaded millennium bug failed to
        ↪  materialise still the people in a number of countries suffered a series
        ↪  of natural disasters that truly were dreadful <EOS>    PP Score = 8.227
        .
        .
        .
        ################## TEST PART #################
        Average Perplexity Score: 16.678
        <SOS> however i would ask you in accordance with the line which is now
        ↪  constantly followed by the european parliament and by the whole of the
        ↪  european community to make representations using the weight of your
        ↪  prestigious office and the institution you represent to the president and
        ↪  to the governor of texas mr bush who has the power to order a stay of
        ↪  execution and to reprieve the condemned person <EOS>    PP Score = 5.833
        <SOS> it seems absolutely disgraceful that we pass legislation and do not
        ↪  adhere to it ourselves <EOS>    PP Score = 10.765
        <SOS> madam president mrs d[U+FFFD]ez gonz[U+FFFD]lez and i had tabled
        ↪  questions on certain opinions of the vice president mrs de palacio which
        ↪  appeared in a spanish newspaper <EOS>    PP Score = 6.264
        .
```

```
.
.
```

2. **LM2:** Tokenization + 4-gram LM + Witten-Bell smoothing (For EuroParl Corpus)

```
################## TRAIN PART #################
Average Perplexity Score: 8.656
<SOS> resumption of the session <EOS>    PP Score = 9.811
<SOS> i declare resumed the session of the european parliament adjourned on
↪  friday <NUM> december 199 and i would like once again to wish you a happy
↪  new year in the hope that you enjoyed a pleasant festive period <EOS>
↪  PP Score = 3.015
<SOS> although as you will have seen the dreaded millennium bug failed to
↪  materialise still the people in a number of countries suffered a series
↪  of natural disasters that truly were dreadful <EOS>    PP Score = 3.461
.
.
.
################## TEST PART #################
Average Perplexity Score: 13.860
<SOS> however i would ask you in accordance with the line which is now
↪  constantly followed by the european parliament and by the whole of the
↪  european community to make representations using the weight of your
↪  prestigious office and the institution you represent to the president and
↪  to the governor of texas mr bush who has the power to order a stay of
↪  execution and to reprieve the condemned person <EOS>    PP Score = 6.829
<SOS> it seems absolutely disgraceful that we pass legislation and do not
↪  adhere to it ourselves <EOS>    PP Score = 16.455
<SOS> madam president mrs d[U+FFFD]ez gonz[U+FFFD]lez and i had tabled
↪  questions on certain opinions of the vice president mrs de palacio which
↪  appeared in a spanish newspaper <EOS>    PP Score = 6.898
.
.
.
```

3. **LM3:** Tokenization + 4-gram LM + Kneyser-Ney smoothing (For Medical Corpus)

```
################## TRAIN PART #################
Average Perplexity Score: 7.680
<SOS> although there are a few isolated reports in the literature suggesting
↪  that sugar beet pollen is highly antigenic hypersensitivity to components
↪  of sugar beet is not a common disease <EOS>    PP Score = 5.821
<SOS> we report a 29 year old man with a history of atopic dermatitis who
↪  developed both contact dermatitis and allergic rhinitis from sugar beet
↪  pollen through his job in a seed nursery <EOS>    PP Score = 5.627
<SOS> establishing nursing science at german universities has become more and
↪  more consolidated <EOS>    PP Score = 10.583
.
.
.
################## TEST PART #################
Average Perplexity Score: 13.860
<SOS> however i would ask you in accordance with the line which is now
↪  constantly followed by the european parliament and by the whole of the
↪  european community to make representations using the weight of your
↪  prestigious office and the institution you represent to the president and
↪  to the governor of texas mr bush who has the power to order a stay of
↪  execution and to reprieve the condemned person <EOS>    PP Score = 6.829
<SOS> it seems absolutely disgraceful that we pass legislation and do not
↪  adhere to it ourselves <EOS>    PP Score = 16.455
<SOS> madam president mrs d[U+FFFD]ez gonz[U+FFFD]lez and i had tabled
↪  questions on certain opinions of the vice president mrs de palacio which
↪  appeared in a spanish newspaper <EOS>    PP Score = 6.898
```

```
.
.
.
```

4. **LM4:** Tokenization + 4-gram LM + Witten-Bell smoothing (For Medical Corpus)

```
################## TRAIN PART #################
Average Perplexity Score: 3.983
<SOS> although there are a few isolated reports in the literature suggesting
↪  that sugar beet pollen is highly antigenic hypersensitivity to components
↪  of sugar beet is not a common disease <EOS>    PP Score = 2.605
<SOS> we report a 29 year old man with a history of atopic dermatitis who
↪  developed both contact dermatitis and allergic rhinitis from sugar beet
↪  pollen through his job in a seed nursery <EOS>    PP Score = 2.515
<SOS> establishing nursing science at german universities has become more and
↪  more consolidated <EOS>    PP Score = 4.147
.
.
.
################## TEST PART #################
Average Perplexity Score: 38.752
<SOS> furthermore changes in cell migration and cytokine production appear to
↪  contribute to the perpetuation of ibd and the postoperative recurrence of
↪  crohn is disease <EOS>    PP Score = 3.762
<SOS> after 3 <NUM> and <NUM> months rontgenograms were taken <EOS>    PP
↪  Score = 12.837
<SOS> 22 patients 12 from the biofix group <NUM> from the ao group operated
↪  two or more years ago were contacted to see if any complications had
↪  occured since they were last seen <EOS>    PP Score = 5.716
.
.
.
```

With Witten-Bell we're getting better *perplexity* score than Kneser-Ney which is expected on any "not-so-big" corpus.

Remarks: For extreme conditions (or edge cases) as detailed below we have assigned probability value $p = \frac{\#\text{ of n-grams with freqency 1}}{\text{size of frequency table of that n-gram}}$, where n can be $4, 3, 2$, or 1. The intuition behind this step is to approximate the count of unseen texts (or tokens) with the counts of seen texts whose frequency is 1 (i.e., it has occured only once) to avoid zero probability and hence it's called *smoothing*.

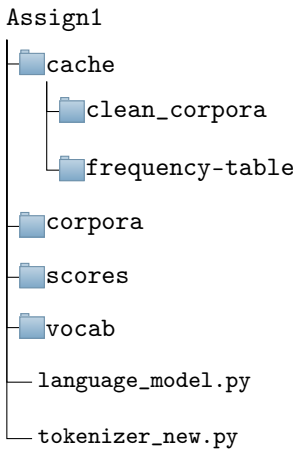
Edge Cases:
Note: `K` stands for "Known" or seen words, while `U` for "unknown" words.

1. **K-K-K-K**: 4 known words but this combination has never occured in training set. For this again we've many subcases. For example cases where *last three* known words have occured together, the recursion will continue and we'll get some probability score. For other cases, at first recursion step only we'll be getting both first term and $\lambda = 0$ which set the probability $= 0$. But to avoid such cases we assign the value p.
2. **K-K-K-U**: In this setting we'll terminate by just checking if n^{th} token in n-gram is in vocabulary or not at the beginning of the function. If no, then p is assigned to it. Else we return to first case.
3. **U-K-K-K** or **K-U-K-K** or cases where atleast one `U` is there, we assign the above mentioned probability value $= p$.

How to execute the code:

Directory Structure:

Extract the `.zip` file to a folder. Say the folder extracted is `Assign1`. Then following should be the directory structure.



1. Once you have the above structure, make sure that you have the following packages installed.

```
numpy>=1.22
re>=2.2
argparse>=1.1
pickle>=4.0
```

2. Now open `terminal` or `command prompt` and run the following command.

```
python3 language_model.py <n_value> <smoothing type> <path to corpus>
```

3. Next, a prompt in terminal will ask to set the name for the `.txt` file for scores we want to save. According to the inputs for `<smoothing type>` and `<path to corpus>`, set the name.

```
if <smoothing type> = k:
    if <path to corpus> = corpora/europarl-corpus.txt:
        set_name = LM1
    elif <path to corpus> = corpora/medical-corpus.txt:
        set_name = LM3
elif <smoothing type> = w:
    if <path to corpus> = corpora/europarl-corpus.txt:
        set_name = LM2
    elif <path to corpus> = corpora/medical-corpus.txt:
        set_name = LM4
```

End of Report