Lab Session-III

(Syntax and Morphological Analysis + POS Tagging using Viterbi Algorithm)

MUSKAN

ASSISTANT PROFESSOR,

CSED, TIET

Regular Expressions for Detecting Word Patterns

- Many linguistic processing tasks involve pattern matching. For example, we can find words ending with ed using endswith('ed') .
- Regular expressions give us a more powerful and flexible method for describing the character patterns we are interested in.
- To use regular expressions in Python, we need to import the re library using: import re .
- We use re.search(p, s) function to check whether the pattern p can be found somewhere inside the string s .
- For example, the words ending with 'ed' can be determined from words of words corpus as :
- wordlist = [w for w in nltk.corpus.words.words('en') if w.islower()]

[w for w in wordlist if re.search('ed$', w)]

Regular Expressions for Detecting Word Patterns (Contd….)

1. . -Wildcard, matches any character
2. ^abc - Matches some pattern abc at the start of a string
3. abc$ -Matches some pattern abc at the end of a string
4. [abc] -Matches one of a set of characters
5. [A-Z0-9] - Matches one of a range of characters
6. ed|ing|s -Matches one of the specified strings (disjunction)
7. * -Zero or more of previous item, e.g., a* , [a-z]* (also known as Kleene Closure)
8. + -One or more of previous item, e.g., a+ , [a-z]+
9. ? -Zero or one of the previous item (i.e., optional), e.g., a? , [a-z]?

Regular Expressions for Detecting Word Patterns (Contd….)

10. {n} - Exactly n repeats where n is a non-negative integer
11. {n,} - At least n repeats
12. {,n} -No more than n repeats
13. {m,n} - At least m and no more than n repeats
14. a(b|c)+ Parentheses that indicate the scope of the operators

Regular Expression Examples

#Collecting words from Treebank words

wsj = sorted(set(nltk.corpus.treebank.words()))

#1. Find all decimal numbers in the wsj list

[w for w in wsj if re.search(‘?’,w)]

#2. Words ending with $ sign

[w for w in wsj if re.search(‘?', w)]

#3. Words with exactly 4 digits

[w for w in wsj if re.search(‘?', w)]

Regular Expression Examples

#Collecting words from Treebank words

wsj = sorted(set(nltk.corpus.treebank.words()))

#1. Find all decimal numbers in the wsj list

[w for w in wsj if re.search('^[0-9]+\.[0-9]+$', w)]

#2. Words ending with $ sign

[w for w in wsj if re.search('^[A-Z]+\$$', w)]

#3. Words with exactly 4 digits

[w for w in wsj if re.search('^[0-9]{4}$', w)]

Regular Expression Examples (Contd…)

#4.words starting with one or number followed by hyphen and 3 to 5 characters

[w for w in wsj if re.search(‘?', w)]

#5.words starting with atleast 5 characters followed by hyphen followed by 2 to 3 letters, hyphen and maximum 6 characters

[w for w in wsj if re.search(‘?', w)]

#6. words ending with ed or ing

[w for w in wsj if re.search(‘?', w)]

Regular Expression Examples (Contd…)

#4.words starting with one or number followed by hyphen and 3 to 5 characters

```
[w for w in wsj if re.search('^[0-9]+-[a-z]{3,5}$', w)]
```

#5.words starting with atleast 5 characters followed by hyphen followed by 2 to 3 letters, hyphen and maximum 6 characters

```
[w for w in wsj if re.search('^[a-z]{5,}-[a-z]{2,3}-[a-z]{,6}$', w)]
```

#6. words ending with ed or ing

```
[w for w in wsj if re.search('(ed|ing)$', w)]
```

Stemming and Lemmatization

Stemming and lemmatization are two methods to convert a word to a non-inflected form. The essence of both stemming and lemmatization is the same: to reduce a word to its most native form. But they differ in how they do it.

- **Stemming** uses a simple mechanism that removes or modifies inflections to form the root word, but the root word may not be a valid word in the language.
- **Lemmatization** also removes or modifies the inflections to form the root word, but the root word is a valid word in the language.

Stemming and Lemmatization (Contd….)

NLTK has several stemmers and lemmatizers (e.g., RegexpStemmer, LancasterStemmer, PorterStemmer, WordNetLemmatizer, RSLPStemmer, and more). There are also many built-in stemmers and lemmatizers you can choose from (see the nltk.stem package).

#Porter Stemmer

```
stemmer = nltk.stem.PorterStemmer()
word = "building"
print("Stem of", word, " is:",stemmer.stem(word))
```

#Lancaster (Paice/Husk) Stemmer

```
stemmer = nltk.stem.LancasterStemmer()
word = "building"
print("Stem of", word, " is:",stemmer.stem(word))
```

Stemming and Lemmatization (Contd….)

#WordNet Lemmatizer

```
lemmatizer = nltk.stem.WordNetLemmatizer()

word = "building"

pos = 'n';

print("Lemmatization of", word, "(" , pos, "):", lemmatizer.lemmatize(word, pos))

pos = 'v';

print("Lemmatization of", word, "(" , pos, "):", lemmatizer.lemmatize(word, pos))
```

Automatic Stemming

- **Feature and Class Method-** compute features among words and then cluster words according to the feature.
- Following code implements automatic stemming in R
- install.packages('stringdist’)
- library(stringdist)
- a<-c('condition','conditions','conditioned','contract','contracts','contracted’)
- b<-stringdistmatrix(a,a,method='jaccard’)
- b<-as.dist(b)
- hc=hclust(b,method='complete’)
- plot(hc)

POS Tagging using HMM

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split
import pprint, time
```

Download libraries

```
#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
?
```

Download libraries

```
#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
print(nltk_data[:2])
```

Check data

```
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)
```

Split Data?

```
# split data into training and validation set in the ratio 80:20
```

Split data

```
# split data into training and validation set in the ratio 80:20

train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state = 101)
```

Extract data in required form

```
# create list of train and test tagged words

train_tagged_words = [?]

test_tagged_words = [?]

print(len(train_tagged_words))

print(len(test_tagged_words))
```

Extract data in required form

```
# create list of train and test tagged words
train_tagged_words = [tup for sent in train_set for tup in sent]
test_tagged_words = [tup for sent in test_set for tup in sent]
print(len(train_tagged_words))
print(len(test_tagged_words))

# check some of the tagged words.
train_tagged_words[:5]
```

Next?

```
#use set datatype to check how many unique tags are present in tr
aining data
```

?

```
# check total words in vocabulary
```

?

Check

```
#use set datatype to check how many unique tags are present in tr
aining data

tags = {tag for word,tag in train_tagged_words}

print(len(tags))

print(tags)


# check total words in vocabulary

vocab = {word for word,tag in train_tagged_words}
```

Emission Probability

```
# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
    tag_list = [pair for pair in train_bag if ?]
    count_tag = ?#total number of times the passed tag occurred in t
rain_bag
    w_given_tag_list = [pair[0] for pair in tag_list if ?]
#now calculate the total number of times the passed word occurred as
 the passed tag.
    count_w_given_tag = ?


    return (count_w_given_tag, count_tag)
```

Emission Probability

```
# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)#total number of times the passed tag occurred in train_bag
    w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
#now calculate the total number of times the passed word occurred as the pas sed tag.
    count_w_given_tag = len(w_given_tag_list)



    return (count_w_given_tag, count_tag)
```

Transition Probability

```
# compute  Transition Probability
def t2_given_t1(t2, t1, train_bag = train_tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[?]==t1 and tags[?] == t2:
            count_t2_t1 =?
    return (count_t2_t1, count_t1)
```

Transition Probability

```
# compute  Transition Probability
def t2_given_t1(t2, t1, train_bag = train_tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2:
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)
```

Creating Matrix

```
# creating t x t transition matrix of tags, t= no of tags
# Matrix(i, j) represents P(jth tag after the ith tag)

tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
for i, t1 in enumerate(list(tags)):
    for j, t2 in enumerate(list(tags)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2
, t1)[1]

print(tags_matrix)
```

Matrix to DF

```
# convert the matrix to a df for better readability

#the table is same as the transition table shown in section 3 of article

tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))

display(tags_df)
```

Viterbi Algorithm

```
def Viterbi(words, train_bag = train_tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))
    for key, word in enumerate(words):
        #initialise list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]
            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = ? * ?
            p.append(state_probability)
        pmax = ? # getting state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

Viterbi Algorithm: Solution

```
def Viterbi(words, train_bag = train_tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))
    for key, word in enumerate(words):
        #initialise list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_p = tags_df.loc['.', tag]
            else:
                transition_p = tags_df.loc[state[-1], tag]
            # compute emission and state probabilities
            emission_p = word_given_tag(words[key], tag)[0]/word_given_tag(words[key], tag)[1]
            state_probability = emission_p * transition_p
            p.append(state_probability)
        pmax = max(p) # getting state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

Lets Test

```
 Let's test our Viterbi algorithm on a few sample sentences of test dataset

random.seed(1234)      #define a random seed to get same sentences when run multiple times

# choose random 10 numbers

rndom = [random.randint(1,len(test_set)) for x in range(10)]

# list of 10 sents on which we test the model

test_run = [test_set[i] for i in rndom]

# list of tagged words

test_run_base = [tup for sent in test_run for tup in sent]

# list of untagged words

test_tagged_words = [tup[0] for sent in test_run for tup in sent]
```

Testing Phase

```
#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end-start
print("Time taken in seconds: ", ?)
# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

Testing Phase

```
#Here We will only test 10 sentences to check the accuracy
#as testing the whole training set takes huge amount of time
start = time.time()
tagged_seq = Viterbi(test_tagged_words)
end = time.time()
difference = end-start
print("Time taken in seconds: ", difference)
# accuracy
check = [i for i, j in zip(tagged_seq, test_run_base) if i == j]
accuracy = len(check)/len(tagged_seq)
print('Viterbi Algorithm Accuracy: ',accuracy*100)
```

Good Luck with Assignments!!!