Natural Language Processing Lab

Lab Manual with Student Lab Record

Developed By

Dr. K. Rajkumar

Roll No	
Name	
Class	**I MSc Data Science**

Department of Data Science
Bishop Heber College (Autonomous)
Tiruchirappalli 620017, INDIA

March 2021

Department of Data Science

Bishop Heber College (Autonomous)

Tiruchirappalli 620017

BONAFIDE CERTIFICATE

Name:____

Reg. No:___________________________________ **Class:**_________________

Course Title ____

Certified that this is the bonafide record of work done by me during **Odd / Even** Semester of **2020 – 2021** and submitted for the Practical Examination on

____

Staff In-Charge **Head of the Department**

Examiners

1.____

2.____

Grade Sheet

Roll No		**Name**	
Year		**Semester**	
		Instructor Name	

Lab	Date	Activity	Grade
1		Understanding Large Text Files	
2		Computing Bigram Frequencies	
3		Computing Document Similarity using VSM	
4		Computing Document Similarity using Word2Vec	
5		Stemming and Lemmatization on Movie Dataset	
6		Spam Filtering using Multinomial Naïve Bayes	
7		Sentiment Analysis on Movie Reviews	
8		Exploring Part of Speech Tagging on Large Text Files	
9		Building Bigram Tagger	
10		Named Entity Recognition on Food Recipes Dataset	
11		Building Parse Trees	
12		Building and Parsing Context Free Grammars	
13		Improving Grammar to Parse Ambiguous Sentences	
14		Word Sense Disambiguation with Improved Lesk	
15		Text Processing using SpaCy	

Preface

This laboratory manual is written to accompany the lab course titled, *Natural Language Processing Lab*. The aim of this laboratory manual is to help students to enhance the understanding of concepts presented in class and to solve problems outlined in the syllabus of a lab course.

The lab exercises have been grouped into weekly activities for a semester. The weekly lab sheets include: input, output, source code and extra credit activities.

Students using these lab sheets should note the following:

1. Fill out your roll no and name in all required places.
2. Read carefully all details of an exercise of a week.
3. Understand the source code which may be a complete code or just a code snippet.
4. The required updates would have been included as comments inside the source code. You need to update them so that your code is ready for execution.
5. Once your code is executable, run your code with the test case inputs and get the results. Verify your obtained output against the expected output.
6. Now, carefully read all extra credit activities, revise your code accordingly, rerun your source code and obtain new outputs.
7. Upon solving all exercises including extra credit activities, approach your lab instructor, demonstrate your experiments and get your grades for the lab.

Final note, attend your weekly lab session with your lab manual without fail. Also, it is your responsibility to keep your lab manual safe as it records the grade you received every week. Comments about the laboratory exercises presented in this Lab manual are welcomed and encouraged. We hope that you will overlook any misspellings, omissions, errors and inconsistencies and report such issues to us. Happy coding!

Dr. K. Rajkumar

Natural Language Processing Lab
Lab1. Understanding Large Text Files

EXERCISE-1

Consider the following text.

```
import nltk
nltk.download('wordnet')
text = "This is Andrew's text, isn't it?"
```

1. How many tokens are there if you use WhitespaceTokenizer?. Print tokens.

```
tokenizer = nltk.tokenize.WhitespaceTokenizer()
tokens = tokenizer.tokenize(text)
print(len(tokens))
print(tokens)
```

2. How many tokens are there if you use TreebankWordTokenizer?. Print tokens.

```
tokenizer = nltk.tokenize.TreebankWordTokenizer()
```

3. How many tokens there are if you use WordPunctTokenizer?. Print tokens.

```
tokenizer = nltk.tokenize.WordPunctTokenizer()
```

EXERCISE-2

1. Open the file: O. Henry's The Gift of the Magi (**gift-of-magi.txt**).

2. Write a Python script to print out the following:

   1. How many word tokens there are
   2. How many word types there are, (word types are a unique set of words)
   3. Top 20 most frequent words and their counts
   4. Words that are at least 10 characters long and their counts
   5. 10+ characters-long words that occur at least twice, sorted from most frequent to least

EXERCISE -3: List Comprehension

STEP-1

Download the document Austen's *Emma* (“**austen-emma.txt"). **Read it in and apply the usual text processing steps, building three objects: etoks (a list of word tokens, all in lowercase), etypes (an alphabetically sorted word type list), and efreq (word frequency distribution).

```
>>> fname = "./data/austen-emma.txt"
>>> f = open(fname, 'r')
>>> etxt = f.read()
>>> f.close()
>>> etxt[-200:]
'e deficiencies, the wishes,\nthe hopes, the confidence, the predictions of the
small band\nof true friends who witnessed the ceremony, were fully answered\nin
the perfect happiness of the union.\n\n\nFINIS\n'
>>> etoks = nltk.word_tokenize(etxt.lower())
>>> etoks[-20:]
['of', 'true', 'friends', 'who', 'witnessed', 'the', 'ceremony', ',', 'were',
'fully', 'answered', 'in', 'the', 'perfect', 'happiness', 'of', 'the', 'union',
'.', 'finis']
>>> len(etoks)
191781
>>> etypes = sorted(set(etoks))
>>> etypes[-10:]
['younger', 'youngest', 'your', 'yours', 'yourself', 'yourself.', 'youth', 'youthful',
'zeal', 'zigzags']
>>> len(etypes)
7944
>>> efreq = nltk.FreqDist(etoks)
>>> efreq['beautiful']
24
```

STEP 2: list-comprehend *Emma*

Now, explore the three objects wlist, efreq, and etypes to answer the following questions. Do NOT use the for loop! Every solution must involve use of LIST COMPREHENSION.

Question 1: Words with prefix and suffix
What are the words that start with 'un' and end in 'able'?

Question 2: Length
How many Emma word types are 15 characters or longer? Exclude hyphenated words.

Question 3: Average word length
What's the average length of all Emma word types?

Question 4: Word frequency
How many Emma word types have a frequency count of 200 or more? How many word types appear only once?

Question 5: Emma words not in wlist
Of the Emma word types, how many of them are not found in our list of ENABLE English words, i.e., wlist?

STEP 3: bigrams in *Emma*

Let's now try out bigrams. Build two objects: e2grams (a list of word bigrams; make sure to cast it as a list) and e2gramfd (a frequency distribution of bigrams) as shown below, and then answer the following questions.

```
>>> e2grams = list(nltk.bigrams(etoks))
>>> e2gramfd = nltk.FreqDist(e2grams)
>>>
```

Question 6: Bigrams
What are the last 10 bigrams?

Question 7: Bigram top frequency
What are the top 20 most frequent bigrams?

Question 8: Bigram frequency count
How many times does the bigram 'so happy' appear?

Question 9: Word following 'so'
What are the words that follow 'so'? What are their frequency counts? (For loop will be easier; see if you can utilize list comprehension for this.)

Question 10: Trigrams
What are the last 10 trigrams? (You can use *nltk.util.ngrams()* method)

Question 11: Trigram top frequency
What are the top 10 most frequent trigrams?

Question 12: Trigram frequency count
How many times does the trigram 'so happy to' appear?

NOTES

NOTES

Natural Language Processing Lab
Lab2. Computing Bigram Frequencies

EXERCISE-1: Process simple bigram data file

STEP 1: OPEN the file, count_2w.txt

When you open it, make sure to specify UTF-8 encoding, otherwise you will see the very last line break. Then, it's business as usual, i.e., reading the file in as a list of lines.

STEP 2: build goog2w_list

First up, build goog2w_list as a list of ((w1, w2), count) tuples. The file this time around is not ordered by frequency, it's ordered alphabetically. That's why we are calling it _list instead of _rank. Here's the process with a mini version:

```
>>> mini = lines[:10]

>>> mini[0]
'0Uplink verified\t523545\n'

>>> mini[0].split()
['0Uplink', 'verified', '523545']

>>> mini_list = []
>>> for m in mini:
        (w1, w2, count) = m.split()
        count = int(count)
        mini_list.append(((w1, w2), count))

>>> mini_list

>>> mini_list[0]
(('0Uplink', 'verified'), 523545)
```

STEP 3: build goog2w_fd

Next, build goog2w_fd as a frequency distribution, implemented as nltk.FreqDist. When finished, it should work like:

```
>>> goog2w_fd[('of', 'the')]
2766332391
>>> goog2w_fd[('so', 'beautiful')]
612472
```

STEP 4: explore

Now explore the two data objects to familiarize yourself with the bigram data. Answer the following questions:

1. What are the top-10 bigrams?
2. What are the top *so*-initial bigrams?
3. Back to those bigrams necessary for computing the probability of the sentence 'She was not afraid.'. Are they all found in this data?
4. Find a bigram that you think should be represented and it is.
5. Find a bigram that you think should be represented but is not.

STEP 5: pickle the data

Pickle `goog2w_list` as 'goog2w_list.pkl', and `goog2w_fd` as 'goog2w_fd.pkl'.

EXERCISE-2: Frequency distribution from Jane Austen Novels

Goto www.nltk.org/nltk_data. Download the zipped archive, **gutenberg.zip**.

This zip file contains 3 novels of Jane Austen (**austen-emma.txt, austen-persuasion.txt, austen-sense.txt**)

Your job is to write a python script that processes the corpora for some basic stats:

A. opens (and later closes) the text file, reads in the string content,
B. builds a list of individual sentences,
C. prints out how many sentences there are,
D. builds a flat tokenized word list and the type list,
E. prints the token and the type counts of this corpus,
F. builds a frequency count dictionary of words,
G. prints the top 50 word types and their counts.

Finally, make one observation about the corpora. It could involve some new code of your own not included above, or it could be based off of A.--F. above.

EXERCISE-3: Bigram Frequencies of Jane Austen Novels

Here, we will take a close look at the bigram frequencies of ***Jane Austen*** novels. We are interested in what types of word bigrams are frequently found in the corpus, and also what types of words are found following the word 'so', and in what probability. Additionally, we will pickle the bigram frequency dictionaries so we can re-use them later.

A. imports necessary modules,
B. opens the text files and reads in the content as text strings,
C. builds the following objects, `a_` for Austen:
    1. `a_toks`: word tokens, all in lowercase
    2. `a_tokfd`: word frequency distribution
    3. `a_bigrams`: word bigrams, cast as a list
    4. `a_bigramfd`: bigram frequency distribution
    5. `a_bigramcfd`: bigram `(w1, w2)` conditional frequency distribution ("CFD"), where `w1` is construed as the condition and `w2` the outcome
D. pickles the bigram CFDs (conditional frequency distributions) using the highest binary protocol: name the file as `austen_bigramcfd.pkl`.
E. answers the following questions by exploring the objects:
    1. How many word tokens and types are there? what is its size?
    2. What are the top 20 most frequent words and their counts?. Draw chart using Matplotlib's plot() method.
    3. What are the top 20 most frequent word bigrams and their counts?, omitting bigrams that contain stopwords
    4. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?
    5. What are the top 20 most frequent word bigrams and their counts, omitting bigrams that contain stopwords?. Draw chart using Matplotlib's plot() method.

6. How many times does the word 'so' occur? What are their relative frequency against the corpus size (= total # of tokens)?
7. What are the top 20 '*so*-initial' bigrams (bigrams that have the word *"so"* as the first word) and their counts?
8. Given the word 'so' as the current word, what is the probability of getting 'much' as the next word?
9. Given the word 'so' as the current word, what is the probability of getting 'will' as the next word?

NOTES

NOTES

Natural Language Processing Lab
Lab3. Computing Document Similarity using VSM

EXERCISE-1: Print TFIDF values

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

docs = [
    "good movie", "not a good movie", "did not like",
    "i like it", "good one" ]

# using default tokenizer in TfidfVectorizer
tfidf = TfidfVectorizer(min_df=2, max_df=0.5, ngram_range=(1, 2))
features = tfidf.fit_transform(docs)
print(features)

# Pretty printing
df = pd.DataFrame(
    features.todense(),
    columns=tfidf.get_feature_names())
print(df)
```

EXERCISE-2:

1. Change the values of **min_df** and **ngram_range** and observe various outputs

EXERCISE-3: Compute Cosine Similarity between 2 Documents

```
from sklearn.metrics.pairwise import linear_kernel

# cosine score between 1st and 2nd doc
doc1 = features[0:1]
doc2 = features[1:2]
score = linear_kernel(doc1, doc2)
print(score)

# cosine score between 1st and all other docs
scores = linear_kernel(doc1, features)
print(scores

# Cosine Similarity for a new doc
query = "I like this good movie"
qfeature = tfidf.transform([query]
scores2 = linear_kernel(doc1, features)
print(scores2)
```

EXERCISE-4: Find Top-N similar documents

Question-1. Consider the following documents and compute TFIDF values

```
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
```

Question-2. Compute cosine similarity between 3rd document (“*the mouse ran away from the house”*) with all other documents. Which is the most similar document?.

Question-3. Find Top-2 similar documents for the 3rd document based on Cosine similarity values.

NOTES

NOTES

Natural Language Processing Lab
Lab4. Computing Document Similarity using Doc2Vec Model

EXERCISE-1

1. Import dependencies

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
from sklearn import utils
```

2. Create dataset

```
data = ["I love machine learning. Its awesome.",
        "I love coding in python",
        "I love building chatbots",
        "they chat amagingly well"]
```

3. Create TaggedDocument

```
tagged_data = [TaggedDocument(words=word_tokenize(d.lower()),
      tags=[str(i)]) for i, d in enumerate(data)]
```

4. Train Model

```
# model parameters
vec_size = 20
alpha = 0.025

# create model
model = Doc2Vec(vector_size=vec_size,
                alpha=alpha,
                min_alpha=0.00025,
                min_count=1,
                dm =1)

# build vocabulary
model.build_vocab(tagged_data)

# shuffle data
tagged_data  = utils.shuffle(tagged_data)

# train Doc2Vec model
model.train(tagged_data,
            total_examples=model.corpus_count,
            epochs=30)

model.save("d2v.model")
print("Model Saved")
```

5. Find Similar documents for the given document

```
from gensim.models.doc2vec import Doc2Vec

model= Doc2Vec.load("d2v.model")

#to find the vector of a document which is not in training data
test_data = word_tokenize("I love chatbots".lower())

v1 = model.infer_vector(test_data)
print("V1_infer", v1)

# to find most similar doc using tags
```

```
similar_doc = model.docvecs.most_similar('1')
print(similar_doc)

# to find vector of doc in training data using tags or
# in other words, printing the vector of document at index 1 in training data

print(model.docvecs['1'])
```

EXERCISE-2

Question1. Train the following documents using Doc2Vec model

```
docs=["the house had a tiny little mouse",
"the cat saw the mouse",
"the mouse ran away from the house",
"the cat finally ate the mouse",
"the end of the mouse story"
]
```

Question2. Find the most similar TWO documents for the query document *"cat stayed in the house".*

NOTES

NOTES

Natural Language Processing Lab
Lab5. Stemming and Lemmatization on Movie Dataset

EXERCISE-1

The file movie.zip contains 20 files about various movies. For each of the files in movies.zip, you will have to do the following:

A. How many sentences in each file?
B. How many tokens in each file?
C. How many tokens excluding stop words in each file?
D. How many unique stems (ie., stemming) in each file? (Use PorterStemmer)
E. How many unique stems (ie., stemming) in each file? (Use LancasterStemmer)
F. How many unique words (ie., lemmatization) in each file? (Use WordNetLemmatizer)
G. Pretty Printing: Print the details of A to E in the following order

```
File Name  Sentences  Tokens  Tokens-Only StemsPorter StemsLancaseter  Lemmas
---------  ---------  ------  ----------- ----------- ---------------  ------
```

EXERCISE-2

In this exercise, you will build your Term-Document Matrix for this movie collection of 20 movies. In order to improve the similarity search experience, you will use only lemmatized terms for creating the matrix.

Step-1

For each movie:
- Tokenize terms and build list of tokens
- Find lemmatized words from the tokens

Step-2
Build Term-Document matrix using TfIdfVectorizer

Step-3
Take vectors of any two movies and compute cosine similarity

EXERCISE-3

Will lemmatized matrix help to achieve better similarity search or not?. Please comment.

NOTES

NOTES

Natural Language Processing Lab
Lab6. Spam Filtering using Multinomial NB

In this lab, you will build Naïve Bayes classifier using SMS data to classify a SMS into spam or not. Once the model is built, it can be used to classify an unknown SMS into spam or ham.

STEPS

1. Open "SMSSpamCollection" file and load into DataFrame. It contains two columns "label" and "text".

2. How many sms messages are there?

3. How many "ham" and "spam" messages?. You need to groupby() label column.

4. Split the dataset into training set and test set (Use 20% of data for testing).

5. Create a function that will remove all punctuation characters and stop words, as below

```
def process_text(msg):
      nopunc =[char for char in msg if char not in string.punctuation]
      nopunc=''.join(nopunc)
      return [word for word in nopunc.split()
            if word.lower() not in stopwords.words('english')]
```

6. Create TfIdfVectorizer as below and perform vectorization on X_train, using fit_perform() method.

```
TfidfVectorizer(use_idf=True,
      analyzer=process_text,
      ngram_range=(1,3),
      min_df = 1,
      stop_words = 'english')
```

7. Create MultinomialNB model and perform training on X_train and y_train using fit() method

8. Predict labels on the test set, using predict() method

9. Print confusion_matrix and classification_report

10. Modify ngram_range=(1,2) and perform Steps 7 to 9.

NOTES

NOTES

Natural Language Processing Lab
Lab7. Sentiment Analysis on Movie Reviews

In this lab, you will build Multinomial Naïve Bayes model for movie reviews from Rotton Tomotto Dataset.

EXERCISE-1

1. Open the file, 'rotten_tomato_train.tsv' and read into a DataFrame

2. Print the basic statistics such as head, shape, describe, and columns

3. How many reviews exist for each sentiment?

EXERCISE-2

1. Extract 200 reviews for each sentiment, store them into a new dataframe and create a smaller dataset. Save this dataframe in a new file, say, **"small_rotten_train.csv"**.

EXERCISE-3

1. Open the file, **"small_rotten_train.csv".**

2. The review text are stored in "Phrase" column. Extract that into a separate DataFrame, say "X".

3. The "sentiment" column is your target, say "y".

4. Perform pre-processing: convert into lower case, remove stop words and lemmatize. The following function will help you.

```
def clean_review(review):
      tokens = review.lower().split()
      filtered_tokens = [lemmatizer.lemmatize(w)
            for w in tokens if w not in stop_words]
      return " ".join(filtered_tokens)
```

5. Apply the above function to X

6. Split X and y for training and testing (Use 20% for testing).

7. Create TfidfVectorizer as below and perfrom vectorization on X_train using fit_perform() method.

```
TfidfVectorizer(min_df=3, max_features=None,
      ngram_range=(1, 2), use_idf=1)
```

8. Create MultinomialNB model and perform training using X_train_lemmartized and y_train.

9. Perform validation on X_test lemmatized and predict output.

10. Print classification_report and accuracy score.

EXERCISE-4

1. Open, 'rotten_tomato_test.tsv' file into dataframe

2. Clean this test data, using the function ***`clean_review(),`*** as before.

3. Build TFIDF values using transform() method.

4. Perform prediction using predict() method.

NOTES

NOTES

Natural Language Processing Lab
Lab8. Exploring POS of Large Text Files

EXERCISE-1

1. Open any movie file from your **movies** sub directory.

2. Tokenize your movie file and print the following

   a. How many sentences in the file?
   b. How many words in the file?
   c. What are the top 10 words and their counts?
   d. How many different POS tags are represented in this file?
   e. What are the top 10 POS tags and their counts?
   f. How many nouns in the file?
   g. How many verbs in the file?
   h. How many adjectives in the file?
   i. How many adverbs in the file?
   j. What is the most frequent adverb?
   k. What is the most frequent adjective?

NOTES

NOTES

Natural Language Processing Lab
Lab9. Building Bigram Tagger

In this lab, you will build a bigram tagger and test it out. We will use the Brown Corpus and its native tagset.

EXERCISE-1

```
import nltk
text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

Output:

```
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

Question: Write down the expansion for CC, RB, …., JJ in the above output.

EXERCISE-2

Type the following lines and load Brown corpus into the list tagsen.

```
from nltk.corpus import brown
tagsen = brown.tagged_sents()
```

STEP 1: Prepare data sets

There are a total of 57,340 POS-tagged sentences in the Brown Corpus. Among them, assign the first 50,000 to your list of training sentences. Then, assign the remaining sentences to your list of testing sentences. The first of your testing sentences should look like this:

```
>>> br_test[0]
[('I', 'PPSS'), ('was', 'BEDZ'), ('loaded', 'VBN'), ('with', 'IN'),
('suds', 'NNS'),
('when', 'WRB'), ('I', 'PPSS'), ('ran', 'VBD'), ('away', 'RB'), (',', ','),
('and', 'CC'),
('I', 'PPSS'), ("haven't", 'HV*'), ('had', 'HVN'), ('a', 'AT'), ('chance',
'NN'),
('to', 'TO'), ('wash', 'VB'), ('it', 'PPO'), ('off', 'RP'), ('.', '.')]
>>>
```

STEP 2: Build a bigram tagger

Following the steps shown in this chapter, build a bigram tagger with two back-off models. The first one on the stack should be a default tagger that assigns `'NN'` by default.

STEP 3: Evaluate

Evaluate your bigram tagger on the test sentences. You should be getting the accuracy score of **0.911.** If not, something went wrong: go back and re-build your tagger.

STEP 4: Explore

Now, explore your tagger to answer the questions below.

1. How big are your training data and testing data? Answer in terms of the number of total words in them.
2. What is the performance of each of the two back-off taggers? How much improvement did you get: (1) going from the default tagger to the unigram tagger, and (2) going from the unigram tagger to the bigram tagger?
3. Recall that 'cold' is ambiguous between `JJ` 'adjective' and `NN` 'singular noun'. Let's explore the word in the training data. The problem with the training data, through, is that it is a list of tagged *sentences*, and it's difficult to get to the tagged words which are one level below:

```
>>> br_train[0]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-
TL'),
('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'),
('investigation',
'NN'), ('of', 'IN'), ("Atlanta's", 'NP$'), ('recent', 'JJ'), ('primary',
'NN'),
('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'AT'),
('evidence',
'NN'), ("''", "''"), ('that', 'CS'), ('any', 'DTI'), ('irregularities',
'NNS'),
('took', 'VBD'), ('place', 'NN'), ('.', '.')]
>>> br_train[1277]          # 1278th sentence
[('``', '``'), ('I', 'PPSS'), ('told', 'VBD'), ('him', 'PPO'), ('who',
'WPS'),
('I', 'PPSS'), ('was', 'BEDZ'), ('and', 'CC'), ('he', 'PPS'), ('was',
'BEDZ'),
('quite', 'QL'), ('cold', 'JJ'), ('.', '.')]
>>> br_train[1277][11]      # 1278th sentence, 12th word
('cold', 'JJ')
>>>
```

4. To be able to compile tagged-word-level statistics, we will need a flat list of tagged words, without them being organized into sentences. How to do this? You can use multi-loop list comprehension to construct it:

```
>>> br_train_flat = [(word, tag) for sent in br_train for (word, tag) in
sent]
                    # [x for innerlist in outerlist for x in innerlist]
>>> br_train_flat[:40]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-
TL'),
('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'),
('investigation',
'NN'), ('of', 'IN'), ("Atlanta's", 'NP$'), ('recent', 'JJ'), ('primary',
'NN'),
('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'AT'),
('evidence',
'NN'), ("''", "''"), ('that', 'CS'), ('any', 'DTI'), ('irregularities',
'NNS'),
('took', 'VBD'), ('place', 'NN'), ('.', '.'), ('The', 'AT'), ('jury',
'NN'),
('further', 'RBR'), ('said', 'VBD'), ('in', 'IN'), ('term-end', 'NN'),
('presentments',
'NNS'), ('that', 'CS'), ('the', 'AT'), ('City', 'NN-TL'), ('Executive',
'JJ-TL'),
('Committee', 'NN-TL'), (',', ','), ('which', 'WDT'), ('had', 'HVD')]
>>> br_train_flat[13]        # 14th word
('election', 'NN')
>>>
```

5. Now, exploring this list of `(word, POS)` pairs from the training data, answer the questions below.
    a. Which is the more likely tag for 'cold' overall?
    b. When the POS tag of the preceding word (call it POS$_{n-1}$) is `AT`, what is the likelihood of 'cold' being a noun? How about it being an adjective?
    c. When POS$_{n-1}$ is `JJ`, what is the likelihood of 'cold' being a noun? How about it being an adjective?
    d. Can you find any POS$_{n-1}$ that favors `NN` over `JJ` for the following word 'cold'?
6. Based on what you found, how is your bigram tagger expected to tag 'cold' in the following sentences?
    a. *I was very cold.*
    b. *I had a cold.*
    c. *I had a severe cold.*
    d. *January was a cold month.*
7. Verify your prediction by having the tagger actually tag the four sentences. What did you find?
8. Have the tagger tag the following sentences, all of which contain the word 'so':
    a. *I failed to do so.*
    b. *I was happy, but so was my enemy.*
    c. *So, how was the exam?*
    d. *The students came in early so they can get good seats.*
    e. *She failed the exam, so she must take it again.*
    f. *That was so incredible.*

   g. *Wow, so incredible.*

9. Examine the tagger's performance on the sentences, focusing on the word 'so'. For each of them, decide if the tagger's output is correct, and explain how the tagger determined the POS tag.
10. Based on what you have observed so far, offer a critique on the bigram tagger. What are its strengths and what are its limitations?

NOTES

NOTES

Natural Language Processing Lab
Lab10. Named Entity Recognition

In this lab, you will extract named entities from the given text file using NLTK. You will also recognize entities based on the regular expression patterns.

EXERCISE-1

Extract all named entities from the following text:

> Sentence1 = "Rajkumar said on Monday that WASHINGTON -- In the wake of a string of abuses by New York police officers in the 1990s, Loretta E. Lynch, the top federal prosecutor in Brooklyn, spoke forcefully about the pain of a broken trust that African-Americans felt and said the responsibility for repairing generations of miscommunication and mistrust fell to law enforcement."

Source Code:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk

tokens = word_tokenize(sentence1)
tags = pos_tag(tokens)
ne_tree = ne_chunk(tags)
print(ne_tree)
```

You can create a pipeline too:

```
ne_tree = ne_chunk(pos_tag(word_tokenize(sentence1)))
```

Question-1

- Count and print the number of PERSON, LOCATION and ORGANIZATION in the given sentence.

Question-2

- Observe the results. Does named entity, “police officers” get recognized?.
- Write a regular expression patter to detect this. You will need nltk.RegexpParser class to define pattern and parse terms to detect patterns.

Question-3

- Does the named entity, “the top federal prosecutor” get recognized?.
- Write a regular expression pattern to detect this.

EXERCISE-2

Extract all named entities from the following text:

> sentence2 = "European authorities fined Google a record **$5.1 billion** on Wednesday for abusing its power in **the mobile phone** market and ordered **the company** to alter its practices"

Question-1

Observe the output. Does your code recognize the NE shown in BOLD?

Write a regular expression that recognizes the entity, “**$5.1 billion”**
Detect and print this

Question-2

Write a regular expression that recognizes the entity, “**the mobile phone”** and similar to this entity such as “**the company”**

EXERCISE-3

In this exercise, you will extract all ingredients from the food recipes text file, food_recipes.txt”. For example, the following text shows one food recipe.

BEEF TENDERLOIN STEAKS WITH SMOKY BACON-BOURBON SAUCE
Serves: 4

1 1/2 cups dry **red wine**
3 cloves **garlic**
1 3/4 cups **beef broth**
1 1/4 cups **chicken broth**
1 1/2 tablespoons **tomato paste**
1 **bay leaf**
1 **sprig thyme**
8 ounces **bacon** cut into 1/4 inch pieces
1 tablespoon **flour**
1 tablespoon **butter**
4 1 inch **rib-eye steaks**
1 tablespoon **bourbon whiskey**

The ingredients are highlighted with BOLD in the above list.

Extract all Named Entities from the text file and display them.

Reference: https://sites.google.com/site/anu3bis/recipes-main.

NOTES

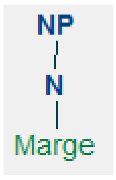
NOTES

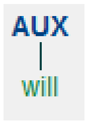
Natural Language Processing Lab
Lab11. Building Parse Trees

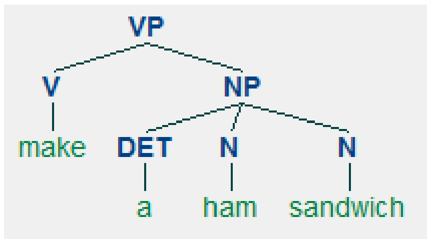
In this lab, you will build parse trees for the given sentences.

EXERCISE-1

Build the following three tree objects as `np`, `aux`, and `vp`.







EXERCISE-2

Create a parse tree for the phrase ***old men and women.*** Is it **well formed sentence** or ambiguous sentence?.

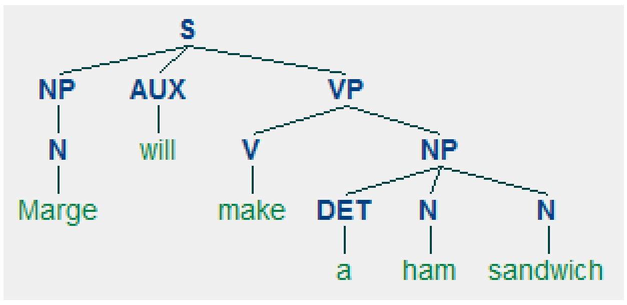
Steps:

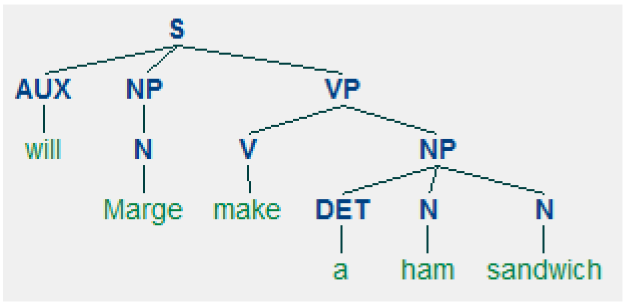
1. Define the grammar (use fromstring() method)
2. Create sentence (as a list of words)
3. Create chart parser
4. Parse and print tree(s)

EXERCISE-3

Using them, build two tree objects, named `s1` and `s2`, for the following sentences. The trees should look exactly like the ones shown below

(s1) Marge will make a ham sandwich
(s2) will Marge make a ham sandwich





EXERCISE-4

Build a tree object named `s3` for the following sentence, using its full-sentence string representation.

(s3) Homer ate the donut on the table

EXERCISE-5

Build tree objects named `s4` and `s5` for the following sentences.

(s4) my old cat died on Tuesday
(s5) children must play in the park with their friends

EXERCISE-6

Once a tree is built, you can extract a list of **context-free rules**, generally called **production rules**, from it using the `.productions()` method. Each CF rule in the list is either lexical, i.e, contains a lexical word on its right-hand side, or not:

```
>>> print(vp)
(VP (V ate) (NP (DET the) (N donut)))
>>> vp_rules = vp.productions()       # list of all CF rules used in the tree
>>> vp_rules
[VP -> V NP, V -> 'ate', NP -> DET N, DET -> 'the', N -> 'donut']
>>> vp_rules[0]
VP -> V NP
>>> vp_rules[1]
V -> 'ate'
>>> vp_rules[0].is_lexical()     # VP -> V NP is not a lexical rule
False
>>> vp_rules[1].is_lexical()     # V -> 'ate' is a lexical rule
True
```

Explore the CF rules of `s5`. Include in your script the answers to the following:

a. How many CF rules are used in `s5`?
b. How many *unique* CF rules are used in `s5`?
c. How many of them are lexical?

NOTES

NOTES

Natural Language Processing Lab
Lab12. Building and Parsing Context Free Grammars

In this lab, you will create Context Free Grammars for the given sentences and parse these sentences using the grammar you wrote.

Please remember the following points, while writing your grammar.

- In the trees and rules, **use lower-case** ('the', 'he') for all words, even at the beginning of a sentence. The only exceptions are the proper names ('Homer', 'Marge', etc.). This simplifies grammar development and parsing.
- For the same reason, **disregard punctuation and symbols** for this assignment.
- For your reference, all the sentences and their tree drawings used in this assignment can be found on this page. Make sure the trees you build matches the tree representation on it.

EXERCISE-1: Build Grammar and Parser

Your job for this part is to develop a context-free grammar (CFG) and a chart parser (as shown in 8.1.2 Ubiquitous ambiguity using the grammar.

1. Using NLTK's `nltk.CFG.fromstring()` method, build a CFG named `grammar1`. The grammar should cover all of the sentences below and their tree structure as presented on this page. The grammar's start symbol should be `'S'`: make sure that an S rule (ex. S -> NP VP) is the very top rule in your list of rules.

   (s6): the big bully punched the tiny nerdy kid after school
   (s7): he gave the book to his sister
   (s8): he gave the book that I had given him t to his sister
   (s9): Homer and Marge are poor but very happy
   (s10): Homer and his friends from work drank and sang in the bar
   (s11): Lisa told her brother that she liked peanut butter very much

2. Once a grammar is built, you can `print` it. Also, you can extract a set of production rules with the `.productions()` method. Unlike the `.productions()` method called on a `Tree` object, the resulting list should be duplicate-free. As before, each rule in the list is a production rule type. A rule has a left-hand side node (the parent node), which you can get to using the `.lhs()` method; the actual string label for the node can be accessed by calling `.symbol()` on the node object.

```
>>> print(grammar3)
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'

>>> grammar3.productions()
[S -> NP VP, NP -> N, VP -> V, N -> 'Homer', V -> 'sleeps']

>>> last_rule = grammar3.productions()[-1]
>>> last_rule
V -> 'sleeps'

>>> last_rule.is_lexical()
True
```

```
>>> last_rule.lhs()                # returns a tree node object
V

>>> last_rule.lhs().symbol()    # returns node label as a string
'V'
```

3. Explore the rules and answer the following questions.
    a. What is the start state of your grammar?
    b. How many CF rules are in your grammar? Is it 71? (It should be.)
    c. How many of them are lexical?
    d. How many VP rules are there? That is, how many rules have 'VP' on the left-hand side of the rule? That is, how many rules are of the `VP -> ...` form?
    e. How many V rules are there? That is, how many rules have 'V' on the left-hand side of the fule? That is, how many rules are of the `V -> ...` form?
4. Using `grammar1`, build a **chart parser**. (Example shown in 8.1.2 Ubiquitous ambiguity.)
5. Using the parser, parse the sentences `s6` -- `s11`. If your `grammar1` is built correctly to cover all of the sentences, the parser should successfully parse all of them.

NOTES

NOTES

Natural Language Processing Lab
Lab13. Improving Grammar to Parse Ambiguous Sentences

In this lab, you will refine the grammar you have built in the previous lab. Because, the grammar does not parse some sentences that are ambiguous.

EXERCISE-1

In this part, you will be updating the grammar and the parser you built in the previous lab.

1. Examine the parser output from the previous lab. Is any of the sentences **ambiguous**, that is, has more than one parse tree? Pick an example and provide an explanation.

2. Have your parser parse this new sentence. It is covered by the grammar, therefore the parser should be able to handle it:

   (s12): Lisa and her friends told Marge that Homer punched the bully in the bar

3. Come up with a sentence of your own that's covered by `grammar1` and have the parser parse it. Are you satisfied with the result?

4. Let's revisit our first three sentences from the previous lab.

   (s1): Marge will make a ham sandwich
   (s2): will Marge make a ham sandwich
   (s3): Homer ate the donut on the table

   As it is, your `grammar1` does not cover them. But we can extend it with the CF rules from the three sentences' trees. Follow the steps below.

   a. From the three sentence trees, create a list of all production rules in them. Turn it into a set, which removes all duplicates. (Hint: use `set()`.)

   b. From it, create a new list called `more_rules`, which consists of CF rules from the three trees *that are not already in `grammar1`*.

   c. Add the additional rules to your `grammar1`'s production rules, using the `.extend()` method.

   d. And then, you have to re-initialize the grammar using the extended production rules (highlighted part). An illustration:

```
>>> print(grammar3)
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'

>>> more_rules
[V -> 'sings', V -> 'drinks']

>>> grammar3.productions().extend(more_rules)
>>> grammar3 = nltk.grammar.CFG(grammar3.start(), grammar3.productions())
>>> print(grammar3)
Grammar with 7 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
    V -> 'sings'
    V -> 'drinks'
>>>
```

e. Now, rebuild your chart parser with the updated `grammar1`. And try parsing the three sentences. It should successfully parse them.

5. Try parsing another sentence of your own that is covered by the newly extended `grammar1`. Are you satisfied with the result?. Also, compare the result with other parsers – Recursive Descent Parser and Shift Reduce Parser.

6. As the final step, pickle your `grammar1` as lab12_grammar.pkl.

NOTES

NOTES

Natural Language Processing Lab
Lab14. Word Sense Disambiguation with Improved Lesk Algorithm

In this lab, you will learn and disambiguate sentences with the correct senses using NLTK.

EXERCISE-1

Consider three examples of the distinct senses that exist for the word "bass":

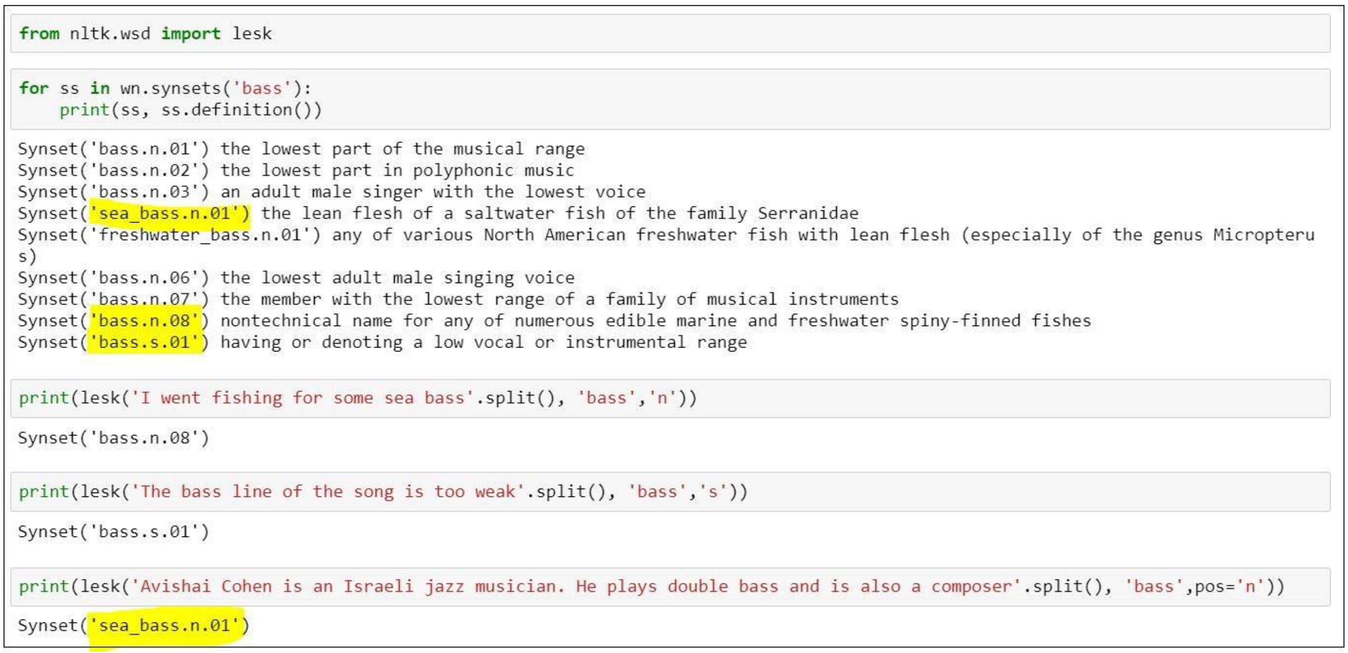
- a type of fish
- tones of low frequency
- a type of instrument

Consider the sentences:

- I went fishing for some sea bass.
- The bass line of the song is too weak.

Lesk algorithm syntax:

```
lesk(context_sentence, ambiguous_word, pos=None, synsets=None)
```



```
from nltk.wsd import lesk
```

```
for ss in wn.synsets('bass'):
    print(ss, ss.definition())
```

```
Synset('bass.n.01') the lowest part of the musical range
Synset('bass.n.02') the lowest part in polyphonic music
Synset('bass.n.03') an adult male singer with the lowest voice
Synset('sea_bass.n.01') the lean flesh of a saltwater fish of the family Serranidae
Synset('freshwater_bass.n.01') any of various North American freshwater fish with lean flesh (especially of the genus Micropteru
s)
Synset('bass.n.06') the lowest adult male singing voice
Synset('bass.n.07') the member with the lowest range of a family of musical instruments
Synset('bass.n.08') nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes
Synset('bass.s.01') having or denoting a low vocal or instrumental range
```

```
print(lesk('I went fishing for some sea bass'.split(), 'bass','n'))
```

```
Synset('bass.n.08')
```

```
print(lesk('The bass line of the song is too weak'.split(), 'bass','s'))
```

```
Synset('bass.s.01')
```

```
print(lesk('Avishai Cohen is an Israeli jazz musician. He plays double bass and is also a composer'.split(), 'bass',pos='n'))
```

```
Synset('sea_bass.n.01')
```

Note: For the third sentence where the bass is in the context of musical instrument, it is estimating the word as Synset('sea_bass.n.01) which is clearly not correct!

EXERCISE-2: Print senses for 'chair'

According to WordNet, how many distinct senses does 'chair' have? What are the hyponyms of 'chair' in its 'chair.n.01' sense? What is its hypernym, and what is its hyper-hypernym?

EXERCISE-3: *Disambiguate the correct senses given the context sentence*

```
from nltk.corpus import wordnet as wn
from nltk.stem import PorterStemmer
from itertools import chain

bank_sents = ['I went to the bank to deposit my money',
'The river bank was full of dead fishes']
```

```
plant_sents = ['The workers at the industrial plant were overworked',
'The plant was no longer bearing flowers']

ps = PorterStemmer()

# define a function my_lesk
def my_lesk(context_sentence, ambiguous_word,
               pos=None, stem=True, hyperhypo=True):

    max_overlaps = 0
    lesk_sense = None
    context_sentence = context_sentence.split()

    for ss in wn.synsets(ambiguous_word):
        # If POS is specified.
        if pos and ss.pos is not pos:
            continue

        lesk_dictionary = []

        # Includes definition.
        defns = ss.definition().split()
        lesk_dictionary +=  defns

        # Includes lemma_names.
        lesk_dictionary += ss.lemma_names()

        # Optional: includes lemma_names of hypernyms and hyponyms.
        if hyperhypo == True:
            hhwords = ss.hypernyms() + ss.hyponyms()
            lesk_dictionary += list(chain(*[w.lemma_names() for w in hhwords] ))

        # Matching exact words causes sparsity,  so lets match stems.
        if stem == True:
            lesk_dictionary = [ps.stem(w) for w in lesk_dictionary]
            context_sentence = [ps.stem(w) for w in context_sentence]

        overlaps = set(lesk_dictionary).intersection(context_sentence)

        if len(overlaps) > max_overlaps:
            lesk_sense = ss
            max_overlaps = len(overlaps)

    return lesk_sense

# evaluate senses
print("Context:", bank_sents[0])
answer = my_lesk(bank_sents[0],'bank')
print("Sense:", answer)
print("Definition:",answer.definition)

print("Context:", bank_sents[1])
answer = my_lesk(bank_sents[1],'bank')
print("Sense:", answer)
print("Definition:", answer.definition)

print("Context:", plant_sents[0])
answer = my_lesk(plant_sents[0],'plant')
print("Sense:", answer)
print("Definition:",answer.definition)
```

EXERCISE-4: Learn further examples for synsets at
https://www.programcreek.com/python/example/91604/nltk.corpus.wordnet.synsets

NOTES

NOTES

Natural Language Processing Lab
Lab15. Text Processing using SpaCy

In this lab session, you will install spacy, displacy and textacy and perform text processing. After completing this lab, you will perform the following NLP tasks.

- Sentence Detection
- Tokenization in spaCy
- Stop Words Removal
- Lemmatization
- Part of Speech Tagging
- Visualization using displaCy
- Rule-Based Matching Using spaCy
- Dependency Parsing Using spaCy
- Navigating the Tree and Subtree
- Shallow Parsing
- Named Entity Recognition

EXERCISES

Question 1. Print the tokens of the string, "welcome all of you for this NLP with spacy course"

Question 2. Create a text file that contains the above string, open that file and print the tokens

Question 3. Consider the following sentences and print each sentence in one line

```
my_text = ('Rajkumar Kannan is a ML developer currently'
...                 ' working for a London-based Edtech'
...                 ' company. He is interested in learning'
...                 ' Natural Language Processing.'
...                 ‵ He keeps organizing local Python meetups'
...                 ' and several internal talks at his workplace.’)
```

Question 4. For the list of strings from **my_text**, print the following for each token:

```
token, token.idx, token.text_with_ws,
token.is_alpha, token.is_punct, token.is_space,
token.shape_, token.is_stop
```

Question 5. Detect and print hyphenated words from **my_text**. For example, `London-based`.

Question 6. Print all stop words defined in SpaCy

Question 7. Remove all stop words and print the rest of tokens from, **my_text**

Question 8. Print all lemma from **my_text**

Question 9. Perform Part of Speech Tagging on my_text and print the following tag informations

```
token, token.tag_, token.pos_, spacy.explain(token.tag_)
```

Question 10. How many NOUN and ADJ are there in **my_text**?. Print them and its count.

Question 11. Visualize POS tags of a sentence, **my_text**, using displaCy

Question 12. Extract and print First Name and Last Name from **my_text** using Matcher.

Question 13. Print the dependency parse tag values for the text, `"Rajkumar is learning piano"`. Also, display dependency parse tree using displaCy.

Question 14. Consider the following string.

```
d_text = ('Sam Peter is a Python developer currently working for a London-based
Fintech company')
```

a. Print the children of `developer`
b. Print the previous neighboring node of `developer`
c. Print the next neighboring node of `developer`
d. Print the all tokens on the left of `developer`
e. Print the tokens on the right of `developer`
f. Print the Print subtree of `developer`

Question 15. Print all Noun Phrases in the text

```
conference_text = ('There is a developer conference happening on 21 July 2020 in
New Delhi.')
```

Question 16. Print all Verb Phrases in the text (you need to install textacy)

```
about_talk_text = ('The talk will introduce reader about Use'
...                     ' cases of Natural Language Processing in'
...                     ' Fintech')
```

Question 17. Print all Named Entities in the text

```
piano_class_text = ('Great Piano Academy is situated'
...     ' in Mayfair or the City of London and has'
...     ' world-class piano instructors.')
```

You will have to print the values such as

```
ent.text, ent.start_char, ent.end_char, ent.label_, spacy.explain(ent.label_)
```

NOTES

NOTES

NOTES

NOTES