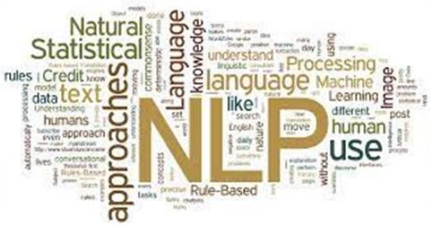
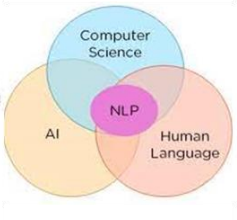
Lab Manual

Natural Language Processing with Python





Dr. G N V G Sirisha

Smt. A. L. Lavanya

Department of Computer Science and Engineering
SRKR Engineering College (A), Bhimavaram, India

This lab manual is intended to aid second-year undergraduate Artificial Intelligence and Machine Learning students in their course *Natural Language Processing with Python Lab* [B20AM2206].

About the author

G N V G Sirisha: She got her PhD, M.Tech. and B.Tech. degrees from Andhra University. Presently she is working as an associate professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India. Her research interests include Data Science, Information Retrieval and Machine Learning.

A L Lavanya: She is currently pursuing (Ph.D) in GIET University, M.Tech. and MCA degrees from Andhra University. Presently she is working as an assistant professor in the Department of Computer Science and Engineering, SRKR Engineering College, Bhimavaram, India.

For private circulation among 2/4 B.Tech.(AI and ML) students.

Preface

Natural Language Processing is a dynamic and rapidly evolving field that sits at the intersection of computer science, linguistics, and artificial intelligence. With the increasing importance of language-based technologies in our daily lives, ranging from chatbots and virtual assistants to language translation and sentiment analysis, NLP has become an essential skill for anyone interested in the world of data science and machine learning.

In this lab, we will leverage the expressive and versatile programming language, Python, along with popular libraries and tools such as NLTK (Natural Language Toolkit), SpaCy, and scikit-learn.

Dr. G N V G Sirisha

Smt. A.L. Lavanya

Evaluation Scheme	
Examination	Marks
Exercise Programs	5
Record	5
Internal Exam	5
External Exam	35

Natural Language Processing with Python Lab

Course Objectives

1. The main objective of the course is to understand the various concepts of natural language processing along with their implementation using Python

Course Outcomes

CO1. Explore natural language processing (NLP) libraries in Python

CO2. Learn various techniques for implementing NLP including parsing & text processing

CO3. Use NLP for text feature engineering, text classification

List of Experiments

S. No.	Experiment
1.	Demonstrate Noise Removal for any textual data and remove regular expression pattern such as hashtag from textual data
2.	Perform lemmatization and stemming using python library nltk
3.	Demonstrate object standardization such as replacing social media slangs from a text
4.	Perform the part of speech tagging on any textual data.
5.	Implement topic modeling using Latent Dirichlet Allocation (LDA) in python.
6.	a. Demonstrate Term Frequency – Inverse Document Frequency (TF – IDF) using python b. Demonstrate word embeddings using word2vec
7.	Implement Text classification using naïve bayes classifier and text blob library
8.	Apply support vector machine for text classification
9.	Convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two text.
10.	Case study 1: Identify the sentiment of tweets In this problem, you are provided with tweet data to predict sentiment on electronicproducts of netizens
11.	Case study 2: Detect hate speech in tweets. The objective of this task is to detect hate speech in tweets. For the sake of simplicity, wesay a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Session # 1

Learning Objective

To demonstrate noise Removal for any textual data and remove regular expression pattern such as hashtag from textual data.

Learning Outcomes

After the completion of this experiment, students will be able to

- Understand different approaches to noise removal
- Develop programs for noise removal using dictionaries and regural expressions

Learning Context

Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.

For example – language stopwords (commonly used words of a language – is, am, the, of, in etc), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.
A general approach for noise removal is to prepare a dictionary of noisy entities, and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

Noise Removal through Regular Expressions

Another approach is to use regular expressions while dealing with special patterns of noise. The following python code removes a regex pattern from the input text:

Important Regular Expression Operators

Operators	Description
.	Matches with any single character except newline '\n'.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.
\d	Matches with digits [0-9] and /D (upper case D) matches with non-digits.
\s	Matches with a single white space character (space, newline, return, tab, form) and \S (upper case S) matches any non-white space character.
\b	boundary between word and non-word and /B is opposite of /b
[..]	Matches any single character in a square bracket and [^..] matches any single character not in square bracket
\	It is used for special meaning characters like \. to match a period or \+ for plus sign.
^ and $	^ and $ match the start or end of the string respectively
{n,m}	Matches at least n and at most m occurrences of preceding expression if we write it as {,m} then it will return at least any minimum occurrence to max m preceding expression.
a| b	Matches either a or b
( )	Groups regular expressions and returns matched text
\t, \n, \r	Matches tab, newline, return

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Demonstrate Noise Removal for any textual data and remove regular expression pattern such as hash tag from textual data

Additional Exercise

a. Create a function to remove all URLs (web links) from a given text. Test the function on a sample text containing URLs.
b. Identify and extract emoticons from the text using regular expressions.
c. Consider various emoticon patterns commonly used in text.

Solutions

```
# Sample code to remove noisy words from a text

noise_list = ["is", "a", "this", "..."]
def _remove_noise(input_text):
    words = input_text.split()
    noise_free_words = [word for word in words if word not in noise_list]
    noise_free_text = " ".join(noise_free_words)
    return noise_free_text

print(_remove_noise("this is a sample text"))
```

Output:

sample text

```
# Sample code to remove a regex pattern
import re
def remove_regex(input_text, regex_pattern):
    result = re.sub(regex_pattern, '', input_text)
    #result=re.sub(r'[\s]+'," ",result)
    return result
regex_pattern = "#[\w]*"
print(remove_regex("remove this #hashtag from analytics vidhya",
regex_pattern))
```

Output:

remove this from analytics vidhya

Output: (With comment line included in the program)

remove this from analytics vidhya

Session # 2

Learning Objective

To Perform lemmatization and stemming using python library nltk.

Learning Outcomes

After the completion of this experiment, students will be able to

- Understand the concepts of lemmatization and stemming
- Apply stemming and lemmatization using Python library nltk

Learning Context

Another type of textual noise is about the multiple representations exhibited by single word.

For example – “play”, “player”, “played”, “plays” and “playing” are the different variations of the word – “play”, Though they mean different but contextually all are similar. The step converts all the disparities of a word into their normalized form (also known as lemma). Normalization is a pivotal step for feature engineering with text as it converts the high dimensional features (N different features) to the low dimensional space (1 feature), which is an ideal ask for any ML model.

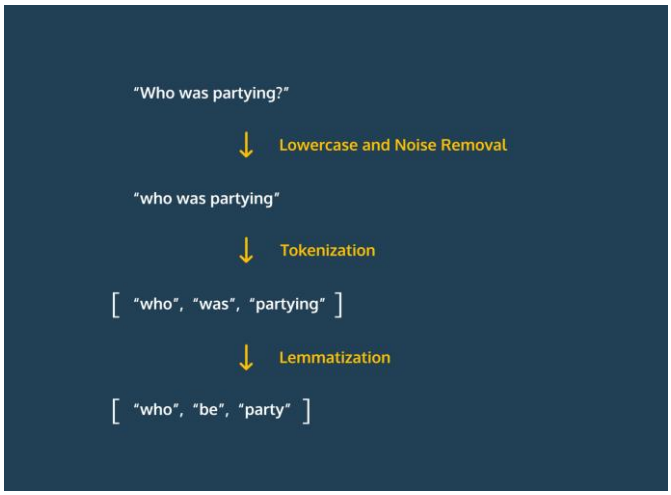
The most common lexicon normalization practices are :

- **Stemming:** Stemming is a rudimentary rule-based process of stripping the suffixes (“ing”, “ly”, “es”, “s” etc) from a word.
- **Lemmatization:** Lemmatization, on the other hand, is an organized & step by step procedure of obtaining the root form of the word, it makes use of

vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

Instructions

In the gif to the right, you can see an example of using noise removal, tokenization, and lemmatization to change the string "Who was partying?" into a list with the words "who", "be", and "party".



Step by Step Process to apply Stemming and Lemmatization

I. Tokenization

In natural language processing, *tokenization* is the text preprocessing task of breaking up text into smaller components of text (known as tokens).

```
from nltk.tokenize import word_tokenize

text = "This is a text to tokenize"
tokenized = word_tokenize(text)

print(tokenized)
# ["This", "is", "a", "text", "to", "tokenize"]
```

II. Text Normalization

In natural language processing, normalization encompasses many text preprocessing tasks including stemming, lemmatization, upper or lowercasing, and stopwords removal.

a. Stemming

In natural language processing, stemming is the text preprocessing normalization task concerned with bluntly removing word affixes (prefixes and suffixes).

If the text is not in tokens, then we need to convert it into tokens. After we have converted strings of text into tokens, we can convert the word tokens into their root form. There are mainly three algorithms for stemming. These are the Porter Stemmer, the Snowball Stemmer and the Lancaster Stemmer. Porter Stemmer is the most common among them.

b. Lemmatization

In natural language processing, *lemmatization* is the text preprocessing normalization task concerned with bringing words down to their root forms. The word "Lemmatization" is itself derived from the base word "Lemma". In Linguistics (a field of study on which NLP is based) a lemma is a meaningful base word or a root word that forms the basis for other words. For example, the lemma of the words "playing" and "played" is play.

c. POS Tagging

POS Tagging (Parts of Speech Tagging) is a process to mark up the words in text format for a particular part of a speech based on its definition and context. Each specific token is assigned a Part of Speech. It is also called grammatical tagging.
Let's learn with a NLTK part of speech example:
Input: Everyting to permit us.
Output: [('Everything', NN), ('to', TO),('permit', VB), ('us', PRP)]

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009
2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/
3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog
4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

1. Perform lemmatization and stemming using python library nltk

Additional Exercise

a. Explore advanced stemming techniques beyond the basic Porter or Lancaster stemmers. Implement and compare the performance of more sophisticated stemmers like Snowball stemmer.

b. Tailor the lemmatization process for a specific domain, such as medical or legal text. Identify domain-specific terms and ensure accurate lemmatization for these terms.

Solutions

```
from nltk.stem import PorterStemmer
tokenized = ["So", "many", "squids", "are", "jumping"]
stemmer = PorterStemmer()
stemmed = [stemmer.stem(token) for token in tokenized]
print(stemmed)
# ['So', 'mani', 'squid', 'are', 'jump']


from nltk.stem.porter import PorterStemmer
from nltk.tokenize import word_tokenize
stemmer = PorterStemmer()
# stem words in the list of tokenised words
def stem_words(text):
    word_tokens = word_tokenize(text)
    stems = [stemmer.stem(word) for word in word_tokens]
    return stems

text = 'data science uses scientific methods algorithms and many types of processes'
stem_words(text)
```

Output:

['data', 'scienc', 'use', 'scientif', 'method', 'algorithm', 'and', 'mani', 'type', 'of', 'process']

```
# Lemmatize with POS Tag
from nltk.corpus import wordnet

def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)
# Return wordnet.NOUN is given key not present in the dictionary

# 1. Init Lemmatizer
lemmatizer = WordNetLemmatizer()

# 2. Lemmatize Single Word with the appropriate POS tag
word = 'feet'
print(lemmatizer.lemmatize(word, get_wordnet_pos(word)))

# 3. Lemmatize a Sentence with the appropriate POS tag
sentence = "The striped bats are hanging on their feet for best"
print([get_wordnet_pos(w) for w in nltk.word_tokenize(sentence)])

print([lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in
nltk.word_tokenize(sentence)])
#> ['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for',
'best']


Foot
['n', 'v', 'n', 'v', 'v', 'n', 'n', 'n', 'n', 'a']

['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best']
```

Session # 3

Learning Objective

To demonstrate object standardization such as replacing social media slangs from a text.

Learning Outcomes

After the completion of this experiment, students will be able to

- Understand the different social media slang
- Develop programs for removing social media slangs from given text

Learning Context

Text data often contains words or phrases which are not present in any standard lexical dictionaries. These pieces are not recognized by search engines and models.

Some of the examples are – acronyms, hashtags with attached words, and colloquial slangs. With the help of regular expressions and manually prepared data dictionaries, this type of noise can be fixed, the code below uses a dictionary lookup method to replace social media slangs from a text.

Object standardization is pre-processing techniques that can be done on abbreviations such as “rt → retweet, dm → direct message”.

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009
2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/
3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog
4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Demonstrate object standardization such as replacing social media slangs from a text

Additional Exercise

a. Develop a function that analyzes a given text and provides statistics on the number of social media slangs replaced. Include information such as the most common slangs and their frequencies.

b. Enhance the slang replacement function to consider the context of words in a sentence. Ensure that only standalone slangs are replaced, and not parts of other words.

c. Extend the object standardization process to include the conversion of common emojis to their textual representations. Replace emojis with corresponding words (e.g., 😂 to "smile").

Solutions

```
lookup_dict = {'rt':'Retweet', 'dm':'direct message', "awsm" : "awesome",
"luv" :"love"}
def _lookup_words(input_text):
    words = input_text.split()
    new_words = []
    for word in words:
        if word.lower() in lookup_dict:
            word = lookup_dict[word.lower()]
        new_words.append(word)
     new_text = " ".join(new_words)
    return new_text

_lookup_words("RT this is a retweeted tweet by Shivam Bansal")

## "Retweet this is a retweeted tweet by Shivam Bansal"

'Retweet this is a retweeted tweet by Shivam Bansal'
```

Session # 4

Learning Objective

To perform the part of speech tagging on any textual data.

Learning Outcomes

After the completion of this experiment, students will be able to

- Define and explain the concept of Part-of-Speech (POS) and its importance in natural language processing.
- Recognize and differentiate between common POS tags such as nouns, verbs, adjectives, adverbs, pronouns, and prepositions.
- Utilize the NLTK (Natural Language Toolkit) library in Python to perform Part-of-Speech tagging on textual data.
- Integrate POS tagging with other natural language processing techniques, such as tokenization, lemmatization, and named entity recognition, to build more advanced language processing pipelines

Learning Context

To analyze preprocessed data, it needs to be converted into features. Depending upon the usage, text features can be constructed using various techniques – Syntactical Parsing, Entities / N-grams / word-based features, Statistical features, and word embeddings. Part-of-speech (POS) tagging is the process of assigning a word to its grammatical category, in order to understand its role within the sentence. Traditional parts of speech are nouns, verbs, adverbs, conjunctions, etc.

Part-of-speech taggers typically take a sequence of words (i.e. a sentence) as input, and provide a list of tuples as output, where each word is associated with the related tag. Part-of-speech tagging is what provides the contextual information that a lemmatiser needs to choose the appropriate lemma.

Part of speech tagging – Apart from the grammar relations, every word in a sentence is also associated with a part of speech (pos) tag (nouns, verbs, adjectives, adverbs etc). The pos tags defines the usage and function of a word in the sentence. Here is a list of all possible pos-tags defined by Pennsylvania university.

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative

23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Part of Speech tagging is used for many important purposes in NLP:

A. Word sense disambiguation: Some language words have multiple meanings according to their usage. For example, in the two sentences below:

I. "Please book my flight for Delhi"

II. "I am going to read this book in the flight"

"Book" is used with different context, however the part of speech tag for both of the cases are different. In sentence I, the word "book" is used as *verb*, while in II it is used as *noun*. (Lesk Algorithm is also used for similar purposes)

B. Improving word-based features: A learning model could learn different contexts of a word when using words as the features, however if the part of speech tag is linked with them, the context is preserved, thus making strong features. For example:

Sentence -"book my flight, I will read this book"

Tokens – ("book", 2), ("my", 1), ("flight", 1), ("I", 1), ("will", 1), ("read", 1), ("this", 1)

Tokens with POS – ("book_VB", 1), ("my_PRP$", 1), ("flight_NN", 1), ("I_PRP", 1), ("will_MD", 1), ("read_VB", 1), ("this_DT", 1), ("book_NN", 1)

C. Normalization and Lemmatization: POS tags are the basis of lemmatization process for converting a word to its base form (lemma).

D. Efficient stopword removal: POS tags are also useful in efficient removal of stopwords.

For example, there are some tags that always define the low frequency / less important words of a language. For example: (**IN** – "within", "upon", "except"), (**CD** – "one","two", "hundred"), (**MD** – "may", "must" etc)

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009
2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/
3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog
4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Perform the part of speech tagging on any textual data.

Additional Exercise

a. Develop a program to integrate parts of speech tagging with named entity recognition

b. Develop programs for parts of speech tagging using other libraries like Spacy, TextBloB, StanfordNLP, Fla, Polyglot and Transformers (HuggingFace)

Solutions

The following programs containing parts of speech tagging using nltk pos_tag and Custom POS Tagging with Penn Treebank Tags

```
from nltk import word_tokenize, pos_tag
text = "I am learning Natural Language Processing on Analytics Vidhya"
tokens = word_tokenize(text)
print(pos_tag(tokens))

Output:

[('I', 'PRP'), ('am', 'VBP'), ('learning', 'VBG'), ('Natural', 'NNP'),
('Language', 'NNP'), ('Processing', 'NNP'), ('on', 'IN'), ('Analytics',
'NNP'), ('Vidhya', 'NNP')]
```

```
# Lemmatize with POS Tag
from nltk.corpus import wordnet

def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)
# Return wordnet.NOUN is given key not present in the dictionary

# 3. Get the appropriate POS tag
sentence = "The striped bats are hanging on their feet for best"
print([get_wordnet_pos(w) for w in nltk.word_tokenize(sentence)])
```

```
['n', 'v', 'n', 'v', 'v', 'n', 'n', 'n', 'n', 'a']
```

Session # 5

Learning Objective

To implement topic modeling using Latent Dirichlet Allocation (LDA) in Python

Learning Outcomes

After the completion of this experiment, students will be able to

- Understand the concept of topic modeling and its significance in uncovering hidden thematic structures within a collection of documents.
- Use Python libraries such as Gensim or Scikit-Learn to implement the LDA algorithm on a corpus of documents.
- Apply topic modeling techniques to real-world scenarios, such as analyzing customer reviews, news articles, or social media content, to extract meaningful insights.

Learning Context

LDA stands for Latent Dirichlet Allocation, which is a probabilistic topic modelling technique used in natural language processing (NLP) and machine learning. It is a generative statistical model that allows for the discovery of underlying topics in a collection of documents. The basic idea behind LDA is that documents are assumed to be composed of a mixture of different topics, and each topic is represented by a distribution of words.

LDA assumes that documents are generated through a two-step process:

Topic assignment: Each document in the collection is assumed to be associated with one or more topics. LDA assigns a distribution of topics to each document, which represents the proportion of topics in that document.

Word assignment: Each word in a document is assumed to be generated from one of the topics assigned to that document.

LDA assigns a specific topic to each word in the document. The goal of LDA is to learn the latent (hidden) topic structure from the observed word occurrences in the documents. It does this by estimating the parameters of the topic and word distributions that best explain the observed data. This process involves iteratively updating the topic and word assignments until a convergence criterion is met.

Steps:

1. Import Libraries
2. Text Pre Processing
   - Converting Text to Lowercase
   - Split Text into Words
   - Remove The Stop Loss Words
   - Removing Punctuation & Special Characters
   - Normalize The Words
3. Converting Text to Numerical Representation
4. Implementation Of LDA
5. Retrieve The Topics
6. Assigning the topic to the documents

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

To implement topic modeling using Latent Dirichlet Allocation (LDA) in Python.

Additional Exercises

1. Use interactive visualization tools (e.g., pyLDAvis) to create an interactive dashboard that allows users to explore and interpret the topics generated by the LDA model.

2. Modify the LDA implementation to include bigrams and trigrams in addition to unigrams. Evaluate the impact on topic coherence and the diversity of topics.

Solutions

1. Import Libraries

```
# for text preprocessing
import re
#import spacy

from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
# import vectorizers
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
# import numpy for matrix operation
import numpy as np
# import LDA from sklearn
from sklearn.decomposition import LatentDirichletAllocation
# to suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

```
D1 = 'I want to watch a movie this weekend.'
D2 =  'I went shopping yesterday. New Zealand won the World Test Championship by beating India by eight wickets at Southampton.'
D3 =  'I don't watch cricket. Netflix and Amazon Prime have very good movies to watch.'
D4 =  'Movies are a nice way to chill however, this time I would like to paint and read some good books. Its been long!'
D5 =  'This blueberry milkshake is so good! Try reading Dr. Joe Dispenzas books. His work is such a game-changer! His books helped to learn so much about how our thoughts impact our biology and how we can all rewire our brains.'
```

```
# combining all the documents into a list:

corpus = [D1, D2, D3, D4, D5]
```

```
import nltk
nltk.download('stopwords')
```

```
nltk.download('wordnet')
nltk.download('omw-1.4')
```

2. Text Preprocessing

Steps to preprocess text data:

1. Convert the text into lowercase
2. Split text into words
3. Remove the stop loss words
4. Remove the Punctuation, any symbols and special characters
5. Normalize the word (I'll be using Lemmatization for normalization)

```
# Apply Preprocessing on the Corpus
# stop loss words
stop = set(stopwords.words('english'))
# punctuation
exclude = set(string.punctuation)
# lemmatization
lemma = WordNetLemmatizer()

# One function for all the steps:
def clean(doc):
    # convert text into lower case + split into words

    #doc = re.sub(r'[\.\?\!\,\:\;\”\']', '', doc)
    #doc = re.sub(r'[\"\#\$\%\&\'\(\)\*\+\,\-
\.\/\:\;\<\=\>\?\@\[\\\]\^\_\`\{\|\}\~\]',"", doc)
    stop_free = " ".join([i for i in doc.lower().split() if i not in
stop])
    print(stop_free)
    # remove any stop words present
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    print(punc_free)
    # remove punctuations + normalize the text
    normalized = " ".join(lemma.lemmatize(word) for word in
    punc_free.split())
    return normalized
# clean data stored in a new list
#clean_corpus = [clean(doc).split() for doc in corpus]
```

3. Convert Text into Numerical Representation

```
# Converting text into numerical representation
tf_idf_vectorizer = TfidfVectorizer( tokenizer=lambda doc:
doc,lowercase=False)
#tf_idf_vectorizer = TfidfVectorizer( lowercase=False)
# Converting text into numerical representation
#cv_vectorizer = CountVectorizer(tokenizer=lambda doc: doc,
lowercase=False)
# Array from TF-IDF Vectorizer
tf_idf_arr = tf_idf_vectorizer.fit_transform(clean_corpus)

# Array from Count Vectorizer
#cv_arr = cv_vectorizer.fit_transform(clean_corpus)
import pandas as pd
df=pd.DataFrame(tf_idf_arr.toarray(),
columns=tf_idf_vectorizer.get_feature_names_out())
print(df)
```

```
     amazon  beating   biology  blueberry      book     brain  championship  \
0  0.000000  0.00000  0.000000   0.000000  0.000000  0.000000       0.00000
1  0.000000  0.27735  0.000000   0.000000  0.000000  0.000000       0.27735
2  0.342983  0.00000  0.000000   0.000000  0.000000  0.000000       0.00000
3  0.000000  0.00000  0.000000   0.000000  0.237416  0.000000       0.00000
4  0.000000  0.00000  0.223316   0.223316  0.360339  0.223316       0.00000

      chill   cricket  dispenzas  ...     watch       way   weekend     went  \
0  0.000000  0.000000   0.000000  ...  0.458270  0.000000  0.568014  0.00000
1  0.000000  0.000000   0.000000  ...  0.000000  0.000000  0.000000  0.27735
2  0.000000  0.342983   0.000000  ...  0.553433  0.000000  0.000000  0.00000
3  0.294271  0.000000   0.000000  ...  0.000000  0.294271  0.000000  0.00000
4  0.000000  0.000000   0.223316  ...  0.000000  0.000000  0.000000  0.00000

    wicket      work    world     would  yesterday  zealand
0  0.00000  0.000000  0.00000  0.000000    0.00000  0.00000
1  0.27735  0.000000  0.27735  0.000000    0.27735  0.27735
2  0.00000  0.000000  0.00000  0.000000    0.00000  0.00000
3  0.00000  0.000000  0.00000  0.294271    0.00000  0.00000
4  0.00000  0.223316  0.00000  0.000000    0.00000  0.00000
```

```
# Creating vocabulary array which will represent all the corpus
vocab_tf_idf = tf_idf_vectorizer.get_feature_names_out()
# get the vocb list
vocab_tf_idf
 # Implementation of LDA:

# Create object for the LDA class
# Inside this class LDA: define the components:
lda_model = LatentDirichletAllocation(n_components = 6, max_iter =
20, random_state = 20)

# fit transform on model on our count_vectorizer : running this
will return our topics
X_topics = lda_model.fit_transform(tf_idf_arr)
print(X_topics)
# .components_ gives us our topic distribution
topic_words = lda_model.components_
#print(topic_words)
```

4.a. Retrieve the Topics

```
#  Define the number of Words that we want to print in every topic
: n_top_words
n_top_words = 5

for i, topic_dist in enumerate(topic_words):

    # np.argsort to sorting an array or a list or the matrix acc to
their values
    sorted_topic_dist = np.argsort(topic_dist)
    #print(sorted_topic_dist)
    # Next, to view the actual words present in those indexes we
can make the use of the vocab created earlier
    print(np.array(vocab_tf_idf))
    print(np.array(vocab_tf_idf)[sorted_topic_dist])
    print("\n ------------------------------")
    topic_words = np.array(vocab_tf_idf)[sorted_topic_dist]
# so using the sorted_topic_indexes we ar extracting the words from
the vocabulary
    # obtaining topics + words
        # this topic_words variable contains the Topics  as well as
    the respective words present in those Topics
    topic_words = topic_words[:-(n_top_words+1):-1]
    print (f"Topic {i+1}", topic_words)
    print("\n ------------------------------")
```

Outputs

Topic 1 ['weekend' 'want' 'watch' 'movie' 'good']

Topic 2 ['watch' 'amazon' 'cricket' 'don't' 'netflix']

Topic 3 ['book' 'zealand' 'test' 'beating' 'world']

Topic 4 ['chill' 'nice' 'would' 'however' 'like']

Topic 5 ['good' 'movie' 'book' 'watch' 'test']

Topic 6 ['good' 'movie' 'book' 'watch' 'test']

4b. Annotating the topics the documents

```
# To view what topics are assigned to the douments:

doc_topic = lda_model.transform(tf_idf_arr)

print(doc_topic)
# iterating over every value till the end value
for n in range(doc_topic.shape[0]):

   # argmax() gives maximum index value
   topic_doc = doc_topic[n].argmax()

   # document is n+1
   print ("Document", n+1, " -- Topic:" ,topic_doc)
```

```
[[0.71953211 0.05629854 0.05603104 0.05607016 0.05603407 0.05603407]
 [0.03620009 0.03619922 0.81899585 0.03619839 0.03620323 0.03620323]
 [0.0449248  0.7761292  0.04472847 0.0447796  0.04471896 0.04471896]
 [0.03652797 0.03650944 0.03649304 0.81757645 0.03644655 0.03644655]
 [0.03142363 0.03146103 0.84279078 0.03147025 0.03142716 0.03142716]]
Document 1  -- Topic: 0
Document 2  -- Topic: 2
Document 3  -- Topic: 1
Document 4  -- Topic: 3
Document 5  -- Topic: 2
```

Session # 6

Learning Objective

a) To demonstrate Term Frequency – Inverse Document Frequency (TF – IDF) using python
b) To demonstrate word embeddings using word2vec

Learning Outcomes

After the completion of this experiment, students will be able to

- Define and explain the concept of Term Frequency-Inverse Document Frequency (TF-IDF) and its role in representing the importance of terms in a document collection.
- Utilize Python libraries such as scikit-learn to implement TF-IDF vectorization on a given corpus of documents.
- Interpret the significance of TF-IDF weighting in terms of assigning higher weights to terms that are important within a specific document while downplaying terms common across multiple documents.
- Define the concept of word embeddings and explain how Word2Vec generates dense vector representations for words based on their contextual usage.
- Utilize Python libraries such as Gensim or scikit-learn to implement Word2Vec on a given corpus and generate word embeddings.

Learning Context

The term frequency-inverse document frequency (tf-idf) is a widely used method for weighting terms in information retrieval and text mining. The tf-idf score measures the importance of a term in a document relative to its importance in the entire corpus. First we split each document in the corpus into a list of words and place them in a set named *words_set* using *union* method. The union() method returns a set that contains all unique items from the specified set(s).

Term Frequency: The term frequency is the number of times a term appears in a document

tf(t,d) = count of t in d / number of words in d

Inverse Document Frequency: Inverse document frequency (idf) is a measure of the rarity of a term in a corpus of documents

idf(t) = log(N / df(t))

where:

- t is a term
- df(t) is the number of documents in the corpus that contain the term t
- N is the total number of documents in the corpus
- log is the natural logarithm

TF-IDF score: The tf-idf score is calculated by multiplying two factors: the term frequency (tf) and the inverse document frequency (idf).

tf-idf(t, d) = tf(t, d) * idf(t)

Word Embedding is a word representation type that allows machine learning algorithms to understand words with similar meanings. It is a language modeling and feature learning technique to map words into vectors of real numbers using neural networks, probabilistic models, or dimension reduction on the word co-occurrence matrix.

Applications of Word Embeddings:

Compute similar words: Word embedding is used to suggest similar words to the word being subjected to the prediction model.

Create a group of related words: It is used for semantic grouping which will group things of similar characteristic together and dissimilar far away

Feature for text classification: Text is mapped into arrays of vectors which is fed to the model for training as well as predict

Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TFIDF) represent each word in a document with a single value, typically a count or a weight, respectively. Word2Vec, on the other hand, represents each word as a vector of values, typically with hundreds of dimensions.

Both Bag of Words and TFiDF ignore the order of words and context of words Word embeddings are developed to overcome this limitation

There are two architectures used by Word2vec:

In CBOW, the current word is predicted using the window of surrounding context windows. For example, if wi-1,wi-2,wi+1,wi+2are given words or context, this model will provide wi

Skip-Gram performs opposite of CBOW which implies that it predicts the given sequence or context from the word. You can reverse the example to understand it. If wi is given, this will predict the context or $w_{i-1}, w_{i-2}, w_{i+1}, w_{i+2}$

we perform dimensionality reduction using PCA and we can plot the points , we can observe that similar words are plotted closely. we found the similarity index between various words in the corpus using 'wv.similarity()' function

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands-on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Demonstrate object standardization such as replacing social media slangs from a text

Additional Exercise

a. Apply TF-IDF to a dataset consisting of short texts (e.g., tweets or headlines). Explore techniques to adapt TF-IDF for short documents.

b. Train a Word2Vec model on a custom dataset, such as a collection of news articles or Wikipedia texts.

c. Create visualizations of Word2Vec embeddings using t-SNE or PCA. Explore the spatial relationships between words in the reduced-dimensional space.

d. Use pre-trained Word2Vec embeddings for sentiment analysis. Fine-tune the embeddings on a sentiment-specific task and assess the model's performance.

Solutions

6. a.

```
import pandas as pd
import numpy as np
corpus = ['data science is one of the most important fields of science',
'this is one of the best data science courses','data scientists analyze data' ]

words_set=set()
for doc in corpus:
    words = doc.split(' ')
    words_set = words_set.union(set(words))

n_docs = len(corpus) #·Number of documents in the corpus
n_words_set = len(words_set) #·Number of unique words in the corpus
df_tf = pd.DataFrame(np.zeros((n_docs, n_words_set)), columns=words_set)

# Compute Term Frequency (TF)
for i in range(n_docs):
    words = corpus[i].split(' ') # Words in the document
    for w in words:
        df_tf[w][i] = df_tf[w][i] + (1 / len(words))

idf = {}
for w in words_set:
    k = 0 # number of documents in the corpus that contain this word
    for i in range(n_docs):
        if w in corpus[i].split():
            k += 1
    idf[w] = np.log10(n_docs / k)

df_tf_idf = df_tf.copy()
for w in words_set:
    for i in range(n_docs):
        df_tf_idf[w][i] = df_tf[w][i] * idf[w]
print(df_tf_idf.sort_index(axis=1))
```

INPUT: corpus mentioned in the program

OUTPUT:

```
    analyze      best   courses  data    fields  important        is      most  \
0  0.00000  0.000000  0.000000   0.0  0.043375   0.043375  0.016008  0.043375
1  0.00000  0.053013  0.053013   0.0  0.000000   0.000000  0.019566  0.000000
2  0.11928  0.000000  0.000000   0.0  0.000000   0.000000  0.000000  0.000000

         of       one   science  scientists       the      this
0  0.032017  0.016008  0.032017     0.00000  0.016008  0.000000
1  0.019566  0.019566  0.019566     0.00000  0.019566  0.053013
2  0.000000  0.000000  0.000000     0.11928  0.000000  0.000000
```

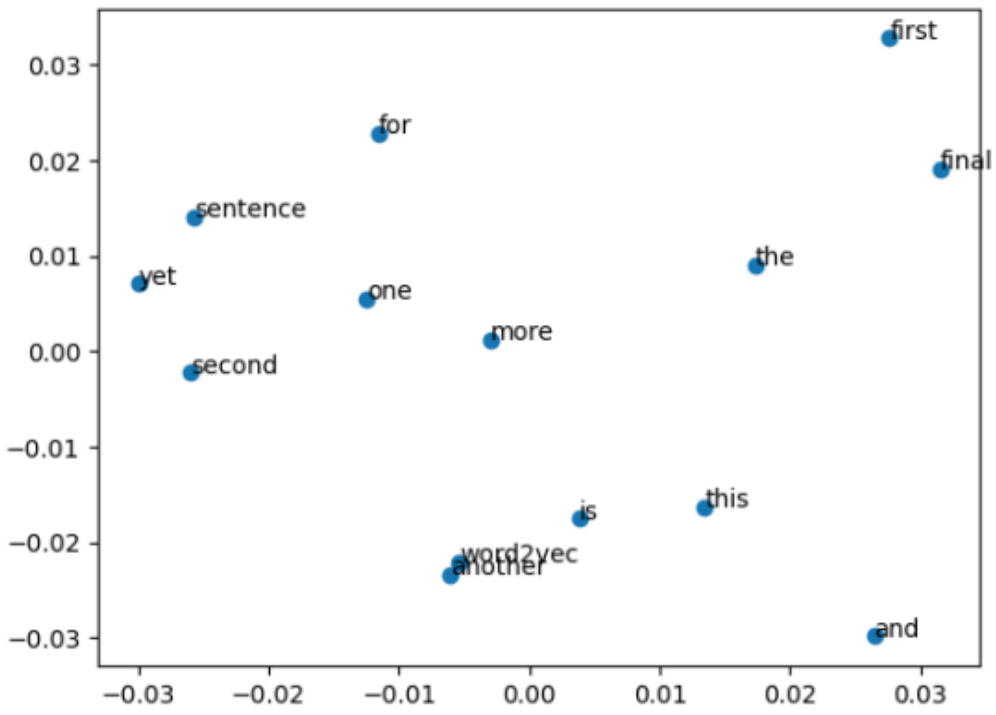
6. b.

```
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
from sklearn.preprocessing import StandardScaler
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
['this', 'is', 'the', 'second', 'sentence'],
['yet', 'another', 'sentence'],
['one', 'more', 'sentence'],
['and', 'the', 'final', 'sentence']]
# train model
model = Word2Vec(sentences, min_count=1)
# fit a 2d PCA model to the vectors
#words = get_unique_words(sentences)
X = model.wv.vectors
#word_vectors_norm = StandardScaler().fit_transform(X)
print(X)
pca = PCA(n_components=2)
result = pca.fit_transform(X)
# create a scatter plot of the projection
pyplot.scatter(result[:, 0], result[:, 1])
words = list(model.wv.index_to_key)
print(words)
for i, word in enumerate(words):
    pyplot.annotate(word, xy=(result[i, 0], result[i, 1]))
pyplot.show()
```

INPUT: custom corpus named ‘sentences’

OUTPUT:

```
[[-5.3622725e-04  2.3643136e-04  5.1033497e-03 … -7.0415605e-03
   9.0145587e-04  6.3925339e-03]
 [-8.6196875e-03  3.6657380e-03  5.1898835e-03 … -2.3915148e-03
  -9.5100943e-03  4.5058788e-03]
 [ 9.4563962e-05  3.0773198e-03 -6.8126451e-03 …  5.1259040e-04
   8.2130842e-03 -7.0190406e-03]
 …
 [ 9.7702928e-03  8.1651136e-03  1.2809718e-03 … -2.9727400e-03
  -4.9318983e-03 -2.3151112e-03]
 [-1.9442164e-03 -5.2675214e-03  9.4471136e-03 …  5.9827138e-03
   6.8153618e-03  7.8225443e-03]
 [-9.5001198e-03  9.5622232e-03 -7.7707553e-03 … -3.1351077e-03
  -6.3388194e-03  9.8700775e-03]]
['sentence', 'the', 'is', 'this', 'final', 'and', 'more', 'one', 'another',
'yet', 'second', 'word2vec', 'for', 'first']
```



```python
from gensim.models import Word2Vec

# define a sample corpus
corpus = [
    ['apple', 'banana', 'orange', 'grape','pear'],
    ['car', 'truck', 'bus', 'bike'],
    ['happy', 'sad', 'angry', 'excited'],
    ['cat', 'dog', 'bird', 'fish']
]

# train a Word2Vec model on the corpus
model = Word2Vec(corpus, vector_size=10, window=15, min_count=1, workers=2)

# check the similarity between similar words
print(model.wv.similarity('apple', 'banana'))
print(model.wv.similarity('bus', 'truck'))
print(model.wv.similarity('happy', 'angry'))
print(model.wv.similarity('cat', 'dog'))
```

INPUT: custom corpus named ‘corpus’

OUTPUT:

```
-0.61994904
0.23240039
-0.09835175
```

Session # 7

Learning Objective

To implement text classification using Naïve Bayes Classifier and Text Blob library

Learning Outcomes

After the completion of this experiment, students will be able to

- Understand the concept of the Naïve Bayes Classifier and its suitability for text classification
- Apply the Naïve Bayes Classifier to real-world scenarios, such as sentiment analysis, spam detection, or topic categorization, showcasing its versatility in different text classification tasks.
- Understand the strengths and limitations of TextBlob for natural language processing.

Learning Context

Text classification is the process of assigning predefined categories or labels to a given text document.

Naive Bayes Classifier is a probabilistic algorithm based on Bayes' theorem that can be used for text classification. It assumes that the presence of a particular feature in a class is independent of the presence of other features in the same class. Naive Bayes Classifier is fast and efficient and works well with high-dimensional data such as text data. Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as follows:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

The Naive Bayes Model

Given a data matrix X and a target vector y, we state our problem as:

$$P(y \mid X) = \frac{P(X \mid y)P(y)}{P(X)}$$

where, y is class variable and X is a dependent feature vector with dimension d i.e. X = (x1,x2,x2, ……. xd), where d is the number of variables/features of the sample. P(y|X) is the probability of observing the class y given the sample X with X = (x1,x2,x2, …. xd), where d is the number of variables/features of the sample.

$$P(y \mid x_1, \ldots, x_d) = \frac{P(y)\prod_{i=1}^{d} P(x_i|y)}{P(x_1)P(x_2)\ldots P(x_d)}$$

The denominator remains constant for the given input so we can remove that term.

$$P(y \mid x_1, \ldots, x_d) \propto P(y)\prod_{i=1}^{d} P(x_i \mid y)$$

Finally, to find the probability of a given sample for all possible values of the class variable y, we just need to find the output with maximum probability:

$$y = \text{argmax}_y\, P(y)\prod_{i=1}^{n} P(x_i \mid y)$$

TextBlob is a Python library built on NLTK for natural language processing (NLP). It simplifies NLP tasks with a user-friendly interface. Key features include:

- Text processing, tagging, extraction, sentiment analysis, and translation.
- Sentiment analysis determines positive, negative, or neutral sentiment in text.

- NaiveBayesClassifier for text classification.
- Easy-to-use API suitable for beginners and experienced NLP practitioners.
- Enables quick and efficient processing and analysis of text data.

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

1. Implement Text classification using naïve bayes classifier and text blob library

Additional Exercise

a. Demonstrate handling of imbalanced classes using Naïve Bayes Classifier

b. Evaluate the performance of the Naïve Bayes classifier using metrics like accuracy, precision, recall, and F1 score.

Solutions

```
import nltk
nltk.download('punkt')
from textblob.classifiers import NaiveBayesClassifier as NBC
from textblob import TextBlob
training_corpus = [
                ('I am exhausted of this work.', 'Class_B'),
                ("I can't cooperate with this", 'Class_B'),
                ('He is my badest enemy!', 'Class_B'),
                ('My management is poor.', 'Class_B'),
                ('I love this burger.', 'Class_A'),
                ('This is an brilliant place!', 'Class_A'),
                ('I feel very good about these dates.', 'Class_A'),
                ('This is my best work.', 'Class_A'),
                ("What an awesome view", 'Class_A'),
                ('I do not like this dish', 'Class_B')]
test_corpus = [
                ("I am not feeling well today.", 'Class_B'),
                ("I feel brilliant!", 'Class_A'),
                ('Gary is a friend of mine.', 'Class_A'),
                ("I can't believe I'm doing this.", 'Class_B'),
                ('The date was good.', 'Class_A'), ('I do not enjoy my job', 'Class_B')]

model = NBC(training_corpus)
print(model.classify("Their codes are amazing."))
print(model.classify("I don't like their computer."))
print(model.accuracy(test_corpus))
```

Output: Class_A
Class_B
0.8333333333333334

Session # 8

Learning Objective

To apply support vector machine for text classification.

Learning Outcomes

After the completion of this experiment, students will be able to

- Explain the basic principles of Support Vector Machines and how they are applied to text classification tasks.
- Utilize Python libraries, such as scikit-learn, to implement a Support Vector Machine model for text classification. Understand the different kernel functions and parameters available in scikit-learn's SVM implementation.

Learning Context

Support vector machines (SVMs) are supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems such as text classification. Text Classification is the process of labeling text data.

SVM finds a hyperplane that creates a boundary between two classes of data to classify them. These algorithms works best on smaller and complex datasets.

To implement SVM for text classification, you can use libraries such as scikit-learn in Python. The steps involved include data preprocessing such as tokenization and word stemming/lemmatization , converting the text into feature vectors using techniques like TF-IDF, splitting the data into training and test sets, and training the SVM classifier on the training data. Finally, you can evaluate the performance of the classifier on the test data.

The classification report consists of the following:

True positive measures the extent to which the model correctly predicts the positive class. False positives occur when the model predicts that an instance belongs to a class that it actually does not. True negatives are the outcomes that the model correctly predicts as negative.

Precision is the ratio between the True Positives and all the Positive. Mathematically:

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Recall is the measure of our model correctly identifying True Positives. Mathematically:

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

F1-score is the Harmonic mean of the Precision and Recall. Mathematically:

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Support values in output of SVM (Support Vector Machine) refer to the degree of confidence with which a given data point is classified into a particular class by the SVM model.

Accuracy is the ratio of the total number of correct predictions and the total number of predictions. Mathematically:

$$\text{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{(True Positive + False Positive + True Negative + False Negative)}}$$

The macro-average is computed using the arithmetic mean (aka unweighted mean) of all the per-class values.
The weighted average is the sum of each singular value multiplied by its corresponding weight.

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009
2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/
3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog
4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

To apply support vector machine for text classification.

Additional Exercise

a. Evaluate the SVM model using Cross-Validation

b. Implement SVM using sklearn, apply different kernel functions and verify which kernel function performs better on IMDB dataset

c. Analyze and interpret feature importance in SVM for text classification. Understand which features contribute more to the decision boundary and classification.

Solutions

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
from sklearn import svm
training_corpus = [
('I am exhausted of this work.', 'Class_B'),
("I can't cooperate with this", 'Class_B'),
('He is my badest enemy!', 'Class_B'),
('My management is poor.', 'Class_B'),
('I love this burger.', 'Class_A'),
('This is an brilliant place!', 'Class_A'),
('I feel very good about these dates.', 'Class_A'),
('This is my best work.', 'Class_A'),
("What an awesome view", 'Class_A'),
('I do not like this dish', 'Class_B')]

test_corpus = [ ("I am not feeling well today.", 'Class_B'),
("I feel brilliant!", 'Class_A'),
('Gary is a friend of mine.', 'Class_A'),
("I can't believe I'm doing this.", 'Class_B'),
('The date was good.', 'Class_A'), ('I do not enjoy my job','Class_B')]
# preparing data for SVM model (using the same training_corpus, test_corpus from naive bayes example)
train_data = []
train_labels = []
for row in training_corpus:
  train_data.append(row[0])
  train_labels.append(row[1])
test_data = []
test_labels = []
for row in test_corpus:
  test_data.append(row[0])
  test_labels.append(row[1])
# Create feature vectors
vectorizer = TfidfVectorizer(min_df=4, max_df=0.9)
# Train the feature vectors
train_vectors = vectorizer.fit_transform(train_data)

# Apply model on test data
test_vectors = vectorizer.transform(test_data)
# Perform classification with SVM, kernel=linear
model = svm.SVC(kernel='linear')
model.fit(train_vectors, train_labels)
prediction = model.predict(test_vectors)
print(prediction)
print(classification_report(test_labels, prediction))
```

OUTPUT:

['Class_A' 'Class_A' 'Class_B' 'Class_B' 'Class_A' 'Class_A']

	precision	recall	F1-score	support
Class A	0.50	0.67	0.57	3
Class B	0.50	0.33	0.40	3
accuracy			0.50	6
Macro avg	0.50	0.50	0.49	6
Weighted avg	0.50	0.50	0.49	6

Session # 9

Learning Objective

To convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two texts.

Learning Outcomes

After the completion of this experiment, students will be able to

- Explain the importance of converting textual data into numerical vectors for various natural language processing (NLP) tasks.
- Implement cosine similarity in Python to measure the similarity between two text vectors.
- Interpret cosine similarity scores and understand how values close to 1 indicate high similarity, while values close to 0 suggest dissimilarity.
- Recognize real-world applications of text vectorization and cosine similarity, such as document similarity, plagiarism detection, and information retrieval.

Learning Context

The process of converting text into vector is called vectorization. Vectorization can be done using many methods. One of the method is "Term Frequency".

Term Frequency (TF) Term frequency is a concept used in natural language preprocessing to describe how often a word appears in a given text document. It is a basic statistic used to represent the importance of a word in the document. The term frequency of a word is calculated by dividing the number of times the word appears in the document by the total number of words in the document.

Formula:

$$Term\ Frequency\ (t) = \frac{Total\ number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. In the context of natural language processing and information retrieval, cosine similarity is commonly used to measure the similarity between two documents or texts.

The Cosine similarity of two documents will range from 0 to 1. If the Cosine similarity score is 1, it means two vectors have the same orientation. The value closer to 0 indicates that the two documents have less similarity.

Formula:

The mathematical equation of Cosine similarity between two non-zero vectors is:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

To convert text to vectors (using term frequency) and apply cosine similarity to provide closeness among two texts.

Additional Exercise

1. Develop a program to visualize cosine similarity using a heat map

2. Extend the exercise to a collection of documents and implement a document clustering algorithm based on cosine similarity. Explore how TF vectors and cosine similarity can be used for unsupervised clustering.

3. Simulate a scenario where the vocabulary dynamically expands as new documents are encountered. Implement a vectorization process that updates the vocabulary and TF vectors on the fly. Discuss the challenges and advantages of dynamic vocabulary expansion.

Solutions

```
import math
from collections import Counter
def get_cosine(vec1, vec2):
    print(vec1.keys())
    print(vec2.keys())
    common = set(vec1.keys()) & set(vec2.keys())
    print(common)
    numerator = sum([vec1[x] * vec2[x] for x in common])
    sum1 = sum([vec1[x]**2 for x in vec1.keys()])
    sum2 = sum([vec2[x]**2 for x in vec2.keys()])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator
def text_to_vector(text):
    words = text.split()
    return Counter(words)
text1 = 'This is an article on analytics vidhya'
text2 = 'article on analytics vidhya is about natural language processing'
vector1 = text_to_vector(text1)
vector2 = text_to_vector(text2)
print(vector1)
print(vector2)
cosine = get_cosine(vector1, vector2)
print(cosine)
```

Output:

```
Counter({'This': 1, 'is': 1, 'an': 1, 'article': 1, 'on': 1, 'analytics': 1, 'vidhya': 1})
Counter({'article': 1, 'on': 1, 'analytics': 1, 'vidhya': 1, 'is': 1, 'about': 1, 'natural': 1, 'language': 1,
'processing': 1})
dict_keys(['This', 'is', 'an', 'article', 'on', 'analytics', 'vidhya'])
dict_keys(['article', 'on', 'analytics', 'vidhya', 'is', 'about', 'natural', 'language', 'processing'])
{'article', 'analytics', 'on', 'is', 'vidhya'}
0.629940788348712
```

Session # 10

Learning Objective

Case study 1: Identify the sentiment of tweets
In this problem, you are provided with tweet data to predict sentiment on electronic products of netizens

Learning Outcomes

After the completion of this experiment, students will be able to

- Recognize the significance of sentiment analysis in understanding public opinions about electronic products.
- Implement text preprocessing techniques to clean and prepare tweet text for sentiment analysis.
- Handle challenges such as removing special characters, handling emojis, and addressing variations in language.

Learning Context

Twitter sentiment analysis identifies negative, and positive emotions within the text of a tweet. It is a text analysis using Natural Language Processing (NLP) and machine learning. It identifies and extracts subjective information from original data, providing a company with a better understanding of the social sentiment of its brand, and product.

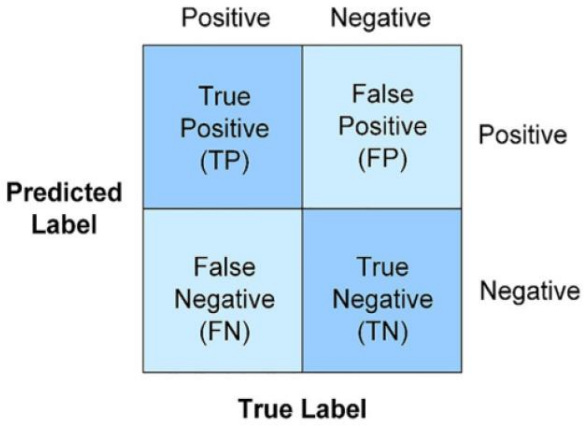
What is Twitter Sentiment Analysis:

A Twitter sentiment analysis is the process of determining the emotional tone behind a series of words, specifically on Twitter. A sentiment analysis tool is an automated technique that extracts meaningful information related to their attitudes, emotions, and opinions.

Steps to perform Twitter Sentiment Analysis:

1. Gather Twitter data:
   To gather information, consider
   a. Historical Tweets: valuable to compare sentiments over different periods.
2. Data Preprocessing: Clean the tweet data by removing any irrelevant information, such as URLs, special characters, and stopwords. Convert the text into numerical features using techniques like TF-IDF or word embeddings.
3. Data Preparation: Split the dataset into training and testing sets. Typically, the dataset is divided into a ratio of 70:30 or 80:20 for training and testing, respectively.
4. Feature Extraction: Extract relevant features from the preprocessed tweet data. This step involves converting the textual data into numerical features that the Random Forest algorithm can understand. Common techniques include Bag-of-Words (BOW)representation, TF-IDF vectorization, or word embeddings (e.g., Word2Vec or GloVe). To perform further analysis we need to transform our data into a format that can be processed by our machine learning models.
   - CountVectorizer does text preprocessing, tokenizing and filtering of stopwords and it builds adictionary of features and transforms documents to feature vectors.
   - TfidfTransformer transforms the above vector by dividing the number of occurrences of each word in a document by the total number of words in the document. These new features are called tf for Term Frequencies.
5. Model Training: Train a Random Forest classifier on the training dataset.Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions. It's well-suited for text classification tasks due to its ability to handle high-dimensional feature spaces and capture complex relationships.

6. Model Evaluation: Evaluate the trained model's performance on the test dataset. Common evaluation metrics for classification tasks include accuracy, precision, recall, and F1-score.



7. Interpretation: Once the model is trained and evaluated, you can interpret the results to gain insights into the sentiment of the tweets.

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Predict sentiment on electronic products of netizens

Solutions

```
from google.colab import drive
drive.mount('/content/drive/')
import numpy as np import
pandas as pd import
seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
import re
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier import numpy as np import
pandas as pd import
seaborn as sns
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize
import re
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
%cd drive
%cd MyDrive/Identify-the-Sentiments-master
# reading dataset
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
# concatenating both datasets
df = pd.concat([train, test])
df.head()
```

	id	label	tweet
0	1	0.0	#fingerprint #Pregnancy Test https://goo.gl/h1…
1	2	0.0	Finally a transparant silicon case ^^ Thanks t…
2	3	0.0	We love this! Would you go? #talk #makememorie…
3	4	0.0	I'm wired I know I'm George I was made that wa…
4	5	1.0	What amazing service! Apple won't even talk to…

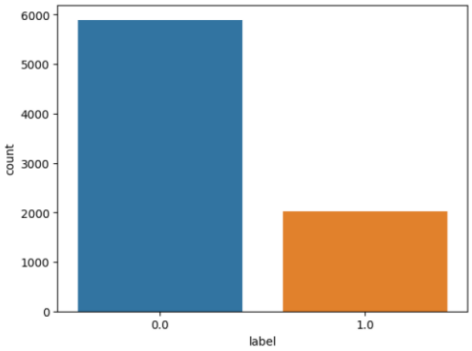
```
# checking for shape
print(train.shape, test.shape, df.shape)
df.describe()
# checking dtypes
df.dtypes
df.isnull().sum()
# Checking unique labels
df['label'].value_counts()
# Ploting unique labels
sns.countplot(x='label', data=df)


# Cleaning Raw tweets
def clean_text(text):
        #remove emails
        text = ' '.join([i for i in text.split() if '@' not in i])
        #remove web address ? Means matching anything to its left \S all non white  space characters
        text = re.sub('http[s]?://\S+', '', text)
        #Filter to allow only alphabets
        text = re.sub(r'[^a-zA-Z\']', ' ', text)
        #Remove Unicode characters
        text = re.sub(r'[^\x00-\x7F]+', '', text)
        #Convert to lowercase to maintain consistency
        text = text.lower()
        #remove double spaces
        text = re.sub('\s+', ' ',text)
return text
df["clean_tweet"] = df.tweet.apply(lambda x: clean_text(x))
```

(7920, 3) (1953, 2) (9873, 3)

```
                id         label
count  9873.000000  7920.000000
mean   4937.000000     0.255808
std    2850.233938     0.436342
min       1.000000     0.000000
25%    2469.000000     0.000000
50%    4937.000000     0.000000
75%    7405.000000     1.000000
max    9873.000000     1.000000
```

```
id          int64
label     float64
tweet      object
dtype: object
```

```
0.0    5894
1.0    2026
Name: label, dtype: int64
```



```
STOP_WORDS = ['a', 'about', 'above', 'after', 'again', 'against', 'all','also', 'am', 'an', 'and','any', 'are',
"aren't", 'as', 'at', 'be', 'because', 'been',↪'before', 'being', 'below','between', 'both', 'but', 'by', 'can',
"can't", 'cannot', 'com',↪'could', "couldn't", 'did',"didn't", 'do', 'does', "doesn't", 'doing', "don't",
'down', 'during', 'each', 'else', 'ever','few', 'for', 'from', 'further', 'get', 'had', "hadn't", 'has',"hasn't",
'have', "haven't", 'having','he', "he'd", "he'll", "he's", 'her', 'here', "here's", 'hers',↪'herself', 'him',
'himself', 'his', 'how',"how's", 'however', 'http', 'i', "i'd", "i'll", "i'm", "i've",'if', 'in', 'into', 'is', "isn't",
'it',"it's", 'its', 'itself', 'just', 'k', "let's", 'like', 'me','more', 'most', "mustn't", 'my', 'myself','no', 'nor',
'not', 'of', 'off', 'on', 'once', 'only', 'or',↪'other', 'otherwise', 'ought', 'our', 'ours',ourselves', 'out', 'over',
'own', 'r', 'same', 'shall',"shan't", 'she', "she'd", "she'll", "she's",'should', "shouldn't", 'since', 'so',
'some', 'such', 'than',↪'that', "that's", 'the', 'their', 'theirs','them', 'themselves', 'then', 'there', "there's",
'these','they', "they'd", "they'll", "they're","they've", 'this', 'those', 'through', 'to', 'too', 'under','until',
'up', 'very', 'was', "wasn't",'we', "we'd", "we'll", "we're", "we've", 'were', "weren't",'what', "what's",
'when', "when's", 'where',"where's", 'which', 'while', 'who', "who's", 'whom', 'why',"why's", 'with',
"won't", 'would', "wouldn't",'www', 'you', "you'd", "you'll", "you're", "you've", 'your','yours',
'yourself', 'yourselves']
```

```
#Remove stopwords from all the tweets
df['cleaned_tweet'] = df['clean_tweet'].apply(lambda x: ' '.join([word for
word in x.split() if word not in (STOP_WORDS)]))
#Adding New feature length of Tweet
df['word_count']=df.cleaned_tweet.str.split().apply(lambda x: len(x))
ndf=df.copy()
ndf = ndf.drop(['tweet','clean_tweet','word_count'], axis = 1)Seperating
Train and Test Set
train_set = ndf[~ndf.label.isnull()]
# Shape
print(train_set.shape,test_set.shape)
# Defining X and Y
X = train_set.drop(['label'], axis=1)
y = train_set.label
# Droping target columns
test_set = test_set.drop(['label'], axis=1)
X=X['cleaned_tweet'].astype(str)
#Train test Split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =
0.3,random_state = 3)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
from sklearn.feature_extraction.text import TfidfTransformer
vect = CountVectorizer()
vect.fit(X_train)
X_train_dtm = vect.transform(X_train)
X_test_dtm = vect.transform(X_test)
model = RandomForestClassifier(n_estimators=200)
model.fit(X_train_dtm,y_train)
rf = model.predict(X_test_dtm)
print("Accuracy:",accuracy_score(y_test,rf)*100,"%")
import itertools
from sklearn.metrics import confusion_matrix
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, rf)
8
cnf_matrix= cnf_matrix / cnf_matrix.astype(np.float).sum(axis=1)
class_names = ['Not Negative','Negative']
# Plot normalized confusion matrix
print(cnf_matrix)
from sklearn.metrics import classification_report
eval_metrics = classification_report(y_test, rf, target_names=class_names)
print(eval_metrics)
```

Accuracy: 86.36363636363636 %

```
[[0.95 0.13]
 [0.14 0.6 ]]
              precision    recall  f1-score   support

Not Negative       0.87      0.95      0.91      1765
    Negative       0.82      0.60      0.69       611

    accuracy                           0.86      2376
   macro avg       0.85      0.78      0.80      2376
weighted avg       0.86      0.86      0.86      2376
```

Session # 11

Learning Objective

Case study 2: Detect hate speech in tweets.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets

Learning Outcomes

After the completion of this experiment, students will be able to

- Define hate speech detection within the context of social media and recognize its significance in curbing offensive and harmful content.
- Utilize machine learning algorithms (e.g., Logistic Regression, Support Vector Machines) for the classification of tweets into categories of racist or sexist hate speech and non-hate speech.

Learning Context

Detecting hate speech in tweets is a challenging task, as it requires understanding the nuances of language and the context in which the words are used. Here are some steps that can help you in detecting hate speech in tweets:

1. Collect a dataset: The first step is to collect a dataset of tweets that contain racist or sexist sentiments. There are several publicly available datasets that you can use, such as the Hate Speech and Offensive Language Identification Dataset (https://github.com/t-davidson/hate-speech-and-offensive-language), which contains tweets labeled as hate speech or not hate speech.
2. Preprocess the data: Preprocess the data by removing noise such as URLs, mentions, and special characters. You can also normalize the data by

converting all the text to lowercase and removing stop words and by applying stemming.

3. Feature extraction: Extract features from the preprocessed data. Some popular feature extraction techniques for text classification include Bag of Words, TF-IDF, and word embeddings.
4. Train a classifier: Use the extracted features to train a classifier. You can use machine learning algorithms such as Naive Bayes, Support Vector Machines, or Logistic Regression.
5. Evaluate the classifier: Evaluate the performance of the classifier using metrics such as accuracy, precision, recall, and F1-score. You can also use techniques such as
6. Cross-validation and grid search to fine-tune the hyper parameters of the classifier.
7. Classify new tweets: Once the classifier is trained, you can use it to classify new tweets as either containing hate speech or not.

Materials & Resources

1. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Steven Bird, Ewan Klein, Edward Loper, Oreilly Publishers, 1st Edition, 2009

2. Ultimate Guide to Understand and Implement Natural Language Processing, https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language- processing-codes-in-python/

3. Identify the sentiments, https://datahack.analyticsvidhya.com/contest/linguipedia-codefest-natural-language-processing-1/?utm_source=ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python&utm_medium=blog

4. Comprehensive Hands on Guide to Twitter Sentiment Analysis, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

Exercise

Develop a model to detect hate speech in tweets

Solution

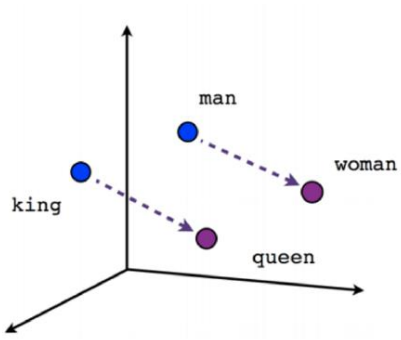
```
from google.colab import drive
drive.mount('/content/gdrive')
%cd gdrive/MyDrive/NLP Lab
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import model_selection, preprocessing,linear_model,metrics
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import ensemble
from sklearn.metrics import roc_auc_score, roc_curve
from xgboost import XGBClassifier
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from textblob import Word
nltk.download('wordnet')
from warnings import filterwarnings
filterwarnings('ignore')
print("\nLIBRARIES WERESUCCESFULLYIMPORTED...")
train_set = pd.read_csv("train.csv", encoding = "utf-8",engine = "python",header = 0)
test_set = pd.read_csv("test.csv", encoding = "utf-8", engine = "python", header = 0)
print("\nDATASETS WERESUCCESFULLYLOADED...")
train_set.head(n = 5)
test_set.head(n = 5)
```

```
print("Train setshape: {} and testsetshape: {}".format(train_set.shape,test_set.shape))
print("Totally thereare {} duplicated valuesintrain_set".format(train_set.duplicated().sum()))
train_set.groupby("label").count().style.background_gradient(cmap = "summer")
train_set["tweet"] = train_set["tweet"].apply(lambda x: " ".join(x.lower() for x in x.split()))
test_set["tweet"] = test_set["tweet"].apply(lambda x: " ".join(x.lower() for x in x.split()))
print("\nCONVERTED SUCCESFULLY...")
# \w--(lowercasew) matches a "word" character:a letter or digit or under bar [a-z A-Z 0-9]
# \s--(lowercases) matches a single white space character
train_set["tweet"] = train_set["tweet"].str.replace('[^\w\s]','')
test_set["tweet"] = test_set["tweet"].str.replace('[^\w\s]','')
print("\n DELETED SUCCESFULLY...")
sw = stopwords.words("english")
train_set['tweet'] = train_set['tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in sw))
test_set['tweet'] = test_set['tweet'].apply(lambda x: " ".join(x for x in x.split() if x not in sw))
print("\nSTOPWORDS DELETEDSUCCESFULLY...")

# Word(word)Creates Word object
train_set['tweet'] = train_set['tweet'].apply(lambda x: " ".join([Word(word).lemmatize() for word in
x.split()]))
test_set['tweet'] = test_set['tweet'].apply(lambda x: " ".join([Word(word).lemmatize() for word in
x.split()]))

train_set = train_set.drop("id", axis = 1)
test_set = test_set.drop("id", axis = 1)
print("\n'ID' COLUMNS DROPPEDSUCCESFULLY...")
x = train_set["tweet"]
y = train_set["label"]
train_x, test_x,train_y,test_y = model_selection.train_test_split(x, y,test_size = 0.20, shuffle = True,
random_state = 11)
print("\nDIVIDED SUCCESFULLY...")

from google.colab import files
from IPython.display import Image
Image('embeddings.png')

vectorizer = CountVectorizer()
vectorizer.fit(train_x)
x_train_count = vectorizer.transform(train_x)
x_test_count = vectorizer.transform(test_x)
x_train_count.toarray()
```

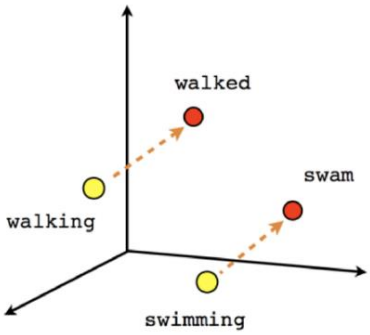
```
tf_idf_word_vectorizer = TfidfVectorizer()
tf_idf_word_vectorizer.fit(train_x)
x_train_tf_idf_word = tf_idf_word_vectorizer.transform(train_x)
x_test_tf_idf_word = tf_idf_word_vectorizer.transform(test_x)
x_train_tf_idf_word.toarray()

log = linear_model.LogisticRegression()
log_model = log.fit(x_train_count, train_y)
accuracy = model_selection.cross_val_score(log_model, x_test_count, test_y, cv = 20)
print(accuracy)
mean = np.mean(accuracy)
print("\nLogistic regressionmodelwith 'count-vectors' method")
print("Accuracy ratio: ", mean)


log = linear_model.LogisticRegression()
log_model = log.fit(x_train_tf_idf_word, train_y)
accuracy = model_selection.cross_val_score(log_model,
x_test_tf_idf_word,
test_y,
cv = 20).mean()
print(accuracy)
mean = np.mean(accuracy)
print("\nLogistic regressionmodelwith 'count-vectors' method")
print("Accuracy ratio: ", mean)


y = train_y
X = x_train_count.astype("float64")
logit_roc_auc = roc_auc_score(y, log_model.predict(X))
fpr, tpr,thresholds = roc_curve(y, log_model.predict_proba(X)[:,1])

plt.figure()
plt.plot(fpr, tpr,label='AUC (area= %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.legend()
plt.xlabel('False PositiveRate')
plt.ylabel('True PositiveRate')
plt.title('ROC')
plt.show()
print(train_x[0])
new_tweet = train_x[0]
new_tweet_features = vectorizer.transform([new_tweet])
log_model.predict(new_tweet_features)
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s…
1	2	0	@user @user thanks for #lyft credit i can't us…
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in …
4	5	0	factsguide: society now #motivation

	id	tweet
0	31963	#studiolife #aislife #requires #passion #dedic…
1	31964	@user #white #supremacists want everyone to s…
2	31965	safe ways to heal your #acne!! #altwaystohe…
3	31966	is the hp and the cursed child book up for res…



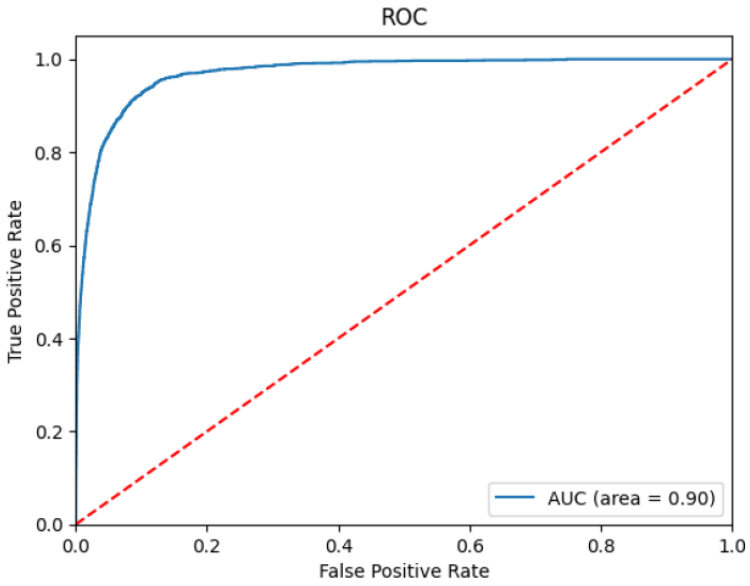
Male-Female



Verb tense

```
[0.953125   0.94375    0.94375    0.95       0.953125   0.953125
 0.94375    0.946875   0.934375   0.965625   0.9375     0.95
 0.946875   0.94984326 0.94670846 0.94670846 0.94670846 0.94357367
 0.94984326 0.95611285]

Logistic regression model with 'count-vectors' method
Accuracy ratio:  0.9480686716300942
```

```
0.9307072884012537

Logistic regression model with 'count-vectors' method
Accuracy ratio:  0.9307072884012537
```



user father dysfunctional selfish drag kid dysfunction run
array([0])