WEEK-2

Aim: Write a program that extracts the words (features) used in a sentence.

Tokenization:

Tokenization is the process of breaking down the given text in natural language processing into the smallest unit in a sentence called a token. Punctuation marks, words, and numbers can be considered tokens.

Example :-
Input: "Welcome to GVP college of engineering”
Output will be: "Welcome” “to” “GVP” “college” “of” “engineering”

Text Blob:

TextBlob is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

Punkt Sentence Tokenizer:

This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used.

Program-1:

```
import textblob
import nltk
nltk.download('punkt')
from textblob import TextBlob
text = "Hello everyone! Welcome to GVP college of engineering."
TextBlob(text).words
```

Output-1:

WordList(['Hello','everyone','Welcome','to','’GVP’,’college’,’of’, ’engineering’])

sent_tokenize: This is for dividing given string into sentences. Here each sentence will be a token.

word_tokenize: This is for dividing given string into words. Here each word will be a token.

Program-2:

```
import nltk

from nltk import sent_tokenize

from nltk import word_tokenize

text = "Hello everyone! Welcome to GVP college of engineering."

tokens_sents = nltk.sent_tokenize(text)

print(tokens_sents)
```

Output-2:

['Hello everyone!', ‘Welcome to GVP college of engineering.']

Stemming:

Stemming is the process of finding the root of words.Stemming is definitely the simpler of the two approaches. With stemming, words are reduced to their word stems. A word stem need not be the same root as a dictionary-based morphological root, it just is an equal to or smaller form of the word.

1. **Overstemming:** Overstemming occurs when words are overtruncated. In such cases, the meaning of the word may be distorted or have no meaning.
2. **Understemming:** Understemming occurs when two words are stemmed from the same root that is not of different stems.

We have two different algorithms in STEMMING.

1. **Porter Stemmer:** This is the Porter stemming algorithm. It follows the algorithm presented in. Porter, M. “An algorithm for suffix stripping.
2. **Snowball Stemmer:** It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the

Porter Stemmer since some issues of it were fixed in this stemmer. Snowball is a small string processing programming language designed for creating stemming algorithms for use in information retrieval. The Snowball compiler translates a Snowball script into program in thread-safe ANSI C, Java, Ada, C#, Go, Javascript, Object Pascal, Python or Rust.

Program-3:

```
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer(language = "english")

word = "civilization"

stemmer.stem(word)

stemmer.stem(“wastes”)
```

Output-3:

'civil'
‘waste’

Lemmatization:

Lemmatization is the process of finding the form of the related word in the dictionary. It is different from Stemming. It involves longer processes to calculate than Stemming. The aim of lemmatization, like stemming, is to reduce inflectional forms to a common base form.

NLTK provides **WordNetLemmatizer** class which is a thin wrapper around the wordnet corpus. This class uses morphy()function to the WordNet CorpusReaderclass to find a lemma.

Wordnet Lemmatizer with NLTK:

Wordnet is an large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most commonly used lemmatizers.

Program-4:

```
import nltk nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("Teachers"))
print(lemmatizer.lemmatize("Doors"))
```

Output-4:

Teacher

Door

Program-5:

```
text = "Let’s lemmatize a simple sentence. We first tokenize the sentence into words using nltk.word_tokenize and then we will call lemmatizer.lemmatize() on each word. "
word_list = nltk.word_tokenize(text) print(word_list)
```

Output-5:

['Let', '’', 's', 'lemmatize', 'a', 'simple', 'sentence', '.', 'We', 'first', 'tokenize', 'the', 'sentence', 'into', 'words', 'using', 'nltk.word_tokenize', 'and', 'then', 'we', 'will', 'call', 'lemmatizer.lemmatize', '(', ')', 'on', 'each', 'word', '.']

Program-6:

```
lemmatized_output = ' .*. '.join([lemmatizer.lemmatize(w) for w in word_list])
print(lemmatized_output)
```

Output-6:

Let .*. ’ .*. s .*. lemmatize .*. a .*. simple .*. sentence .*. . .*. We .*. first .*. tokenize .*. the .*. sentence .*. into .*. word .*. using .*. nltk.word_tokenize .*. and .*. then .*. we .*. will .*. call .*. lemmatizer.lemmatize .*. (.*.) .*. on .*. each .*. word .*. .

Program-7:

```
from textblob import TextBlob, Word
word = 'stripes'
w = Word(word)
```

```
w.lemmatize()

text = "The striped bats are hanging on their feet for best"

sent = TextBlob(text)

" ". join([w.lemmatize() for w in sent.words])
```

Output-7:

'stripe'

'The striped bat are hanging on their foot for best'

Part of Speech Tagging#

Part of Speech Tagging (POS-Tag) is the labeling of the words in a text according to their word types (noun, adjective, adverb, verb, etc.). It is a process of converting a sentence to forms, list of words, list of tuples The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

POS tagging is a supervised learning solution that uses features like the previous word, next word, is first letter capitalized etc. NLTK has a function to get pos tags and it works after tokenization process. **Averaged_perceptron_tagger:**

The Perceptron tagger, also known as Averaged Perceptron Tagger is ported from TextBlob Perceptron Tagger into NLTK and is implemented originally by Matthew Honnibal.

Program-8:

```
import nltk

nltk.download('averaged_perceptron_tagger')

from nltk import word_tokenize

text = "The striped bats are hanging on their feet for best"

tokens = nltk.word_tokenize(text)

print("Parts of Speech: ",nltk.pos_tag(tokens))
```

Output-8:

Parts of Speech: [('The', 'DT'), ('striped', 'JJ'), ('bats', 'NNS'), ('are', 'VBP'), ('hanging', 'VBG'), ('on', 'IN'), ('their', 'PRP$'), ('feet', 'NNS'), ('for', 'IN'), ('best', 'JJS')]

WEEK-3

Aim: Write a program for edge detection to extract edge based features from a sample image.

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. `cv2.imread()` method loads an image from the specified file. If the image cannot be read then this method returns an empty matrix.

Syntax: *cv2.imread(path, flag)*

Program-1:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import pylab
pylab.rcParams['figure.figsize']=[10.0,8.0]
input_image=cv2.imread('hallphoto3.jpeg')
print("Image dimension values")
print(input_image)
print("Matrix size")
print(input_image.size)
print("shape")
print(input_image.shape)
```

Output-1:

```
Image dimension values
[[[155 174 181]
  [155 174 181]
  [154 173 180]
  ...
  [146 168 179]
  [146 168 179]
  [146 168 179]]

 [[155 174 181]
  [155 174 181]
  [154 173 180]
  ...
  [146 168 179]
```

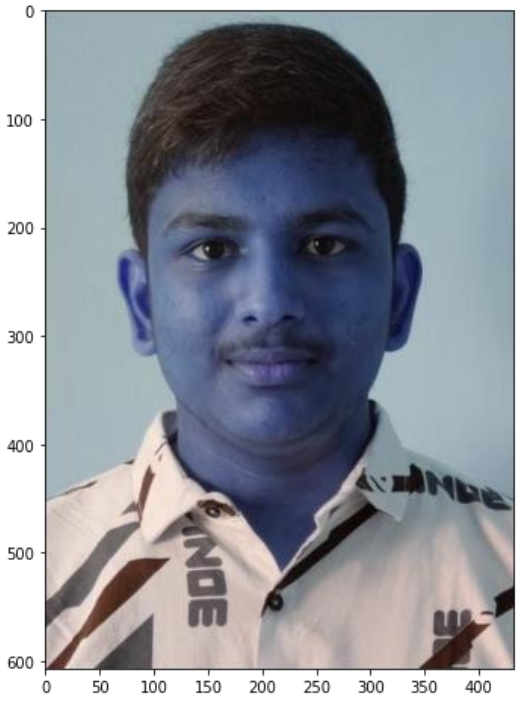
```
  [146 168 179]
  [146 168 179]]

 [[155 174 181]
  [155 174 181]
  [154 173 180]
  ...
  [146 168 179]
  [146 168 179]
  [146 168 179]]

 ...

 [[209 193 187]
  [211 195 189]
  [196 176 171]
  ...
  [197 185 183]
  [185 177 177]
  [177 172 173]]

 [[213 197 191]
  [201 182 177]
  [161 139 134]
  ...
  [199 187 185]
  [188 180 180]
  [180 175 176]]

 [[214 195 190]
  [177 158 153]
  [128 106 101]
  ...
  [202 190 186]
  [193 184 181]
  [183 178 179]]]
Matrix size
788493
shape
(607, 433, 3)
```

Program-2:

```
print("Displaying image")
print(plt.imshow(input_image))
```

Output-2:

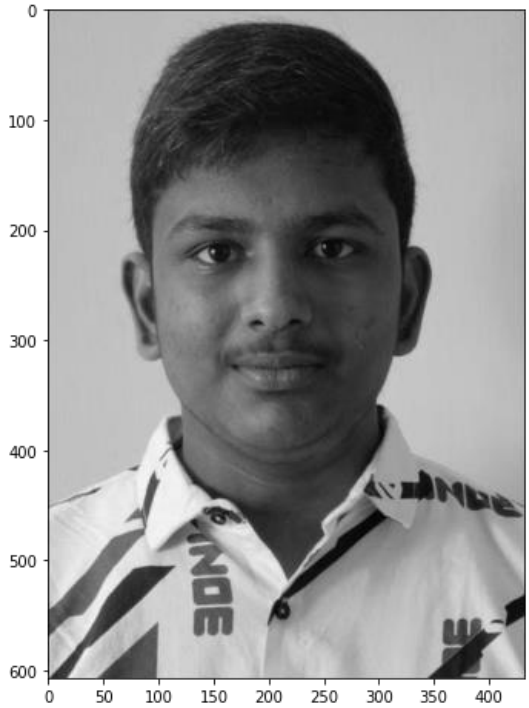
```
---Displaying image---
AxesImage(80,52.8;496x369.6)
```



Program-3:

```
b,g,r=cv2.split(input_image)
plt.imshow(g,cmap='gray')
```

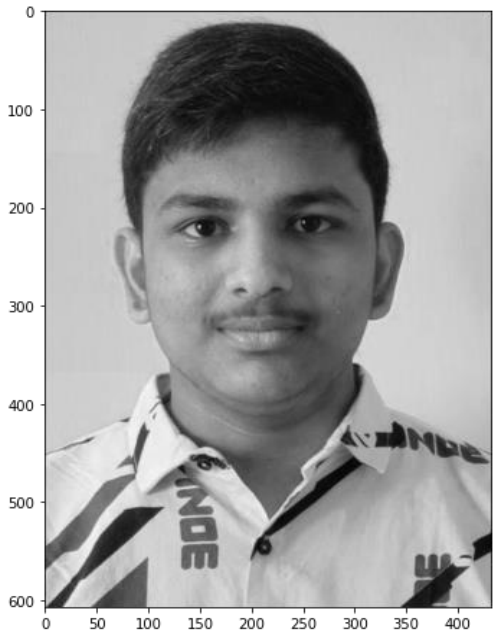
Output-3:



Program-4:

```
b,g,r=cv2.split(input_image)
plt.imshow(r,cmap='gray')
```

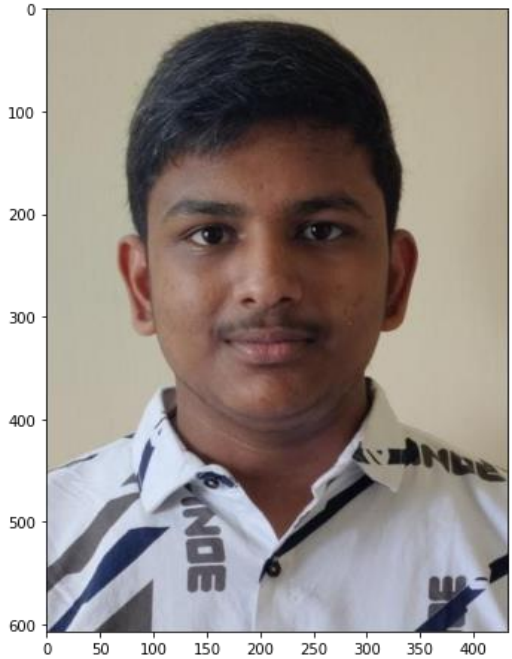
Output-4:



Program-5:

```
merged=cv2.merge([r,g,b])
plt.imshow(merged)
```

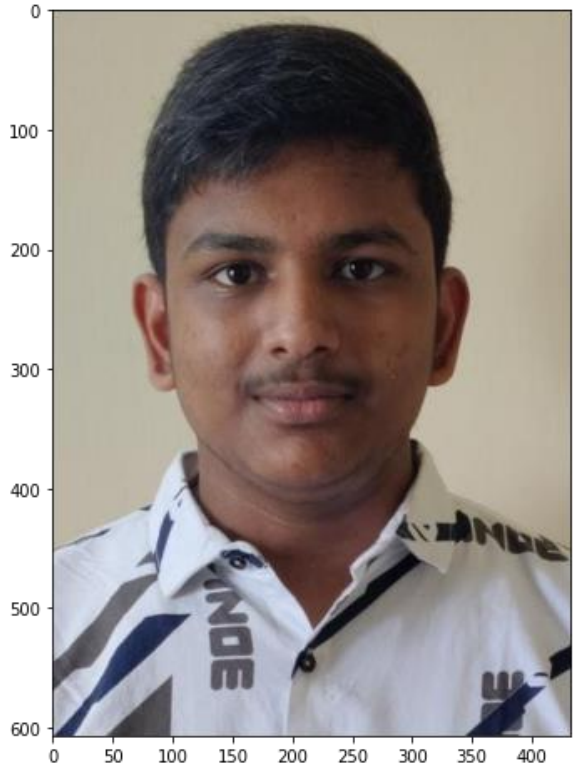
Output-5:



Program-6:

```
print("---BGR TO RGB---")
opencv_merged=cv2.cvtColor(input_image,cv2.COLOR_BGR2RGB)
plt.imshow(opencv_merged)
```

Output-6:



Program-7:

```
print("---print pixels---")
pixel=input_image[100,100]
print(pixel)
print("---print modified pixels---")
input_image[100,100]=[0,0,0]
pixel_new=input_image[100,10]
print(pixel_new)
```

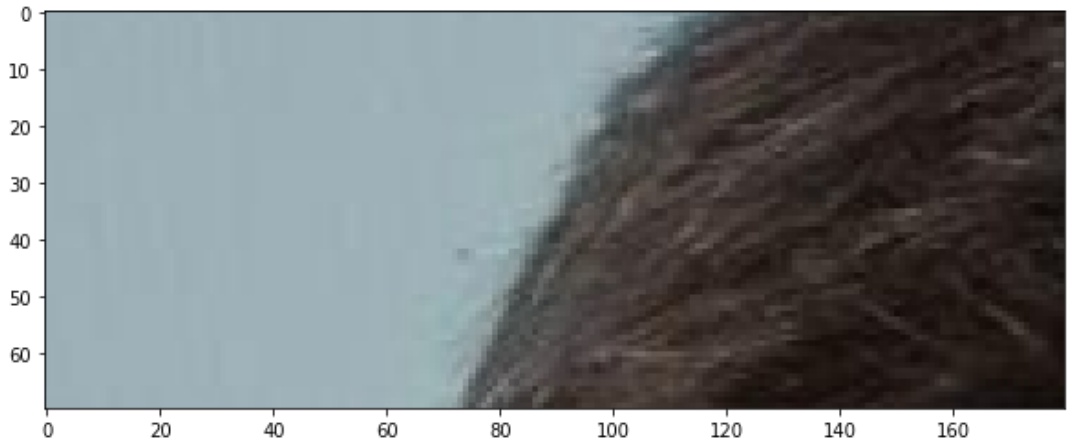
Output-7:

```
---print pixels---
[63 54 50]
---print modified pixels---
[0 0 0]
```

Program-8:

```
print("---Crop image---")
input_image2=input_image[30:100,10:190] plt.imshow(input_image2)
```

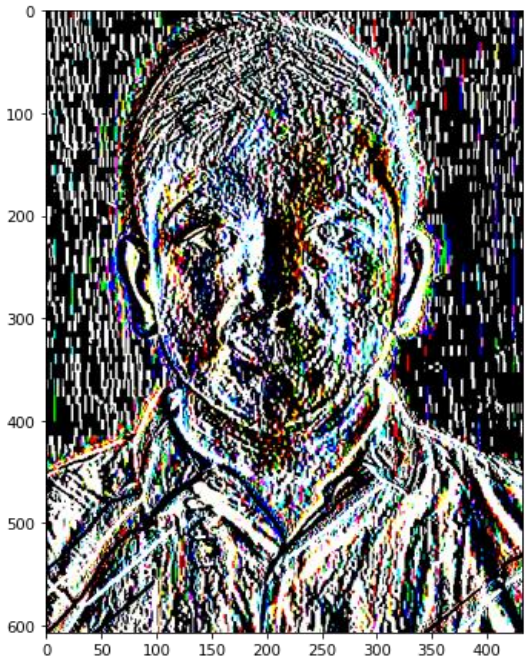
Output-8:



Program-9:

```
s1=cv2.Sobel(input_image,cv2.CV_64F,1,0,ksize5)
plt.imshow(s1)
print(s1.size)
 print(s1.shape)
```

Output-9:

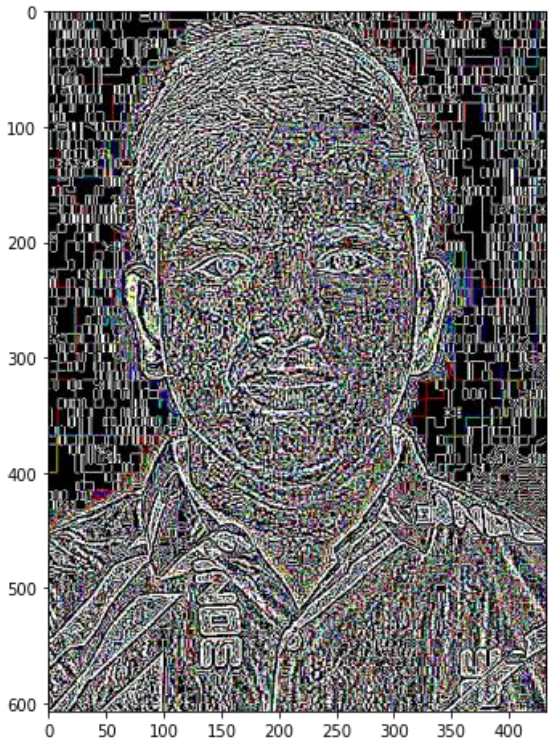


```
788493
(607, 433, 3)
```

Program-10:

```
l1=cv2.Laplacian(input_image,cv2.CV_64F)
plt.imshow(l1)
```

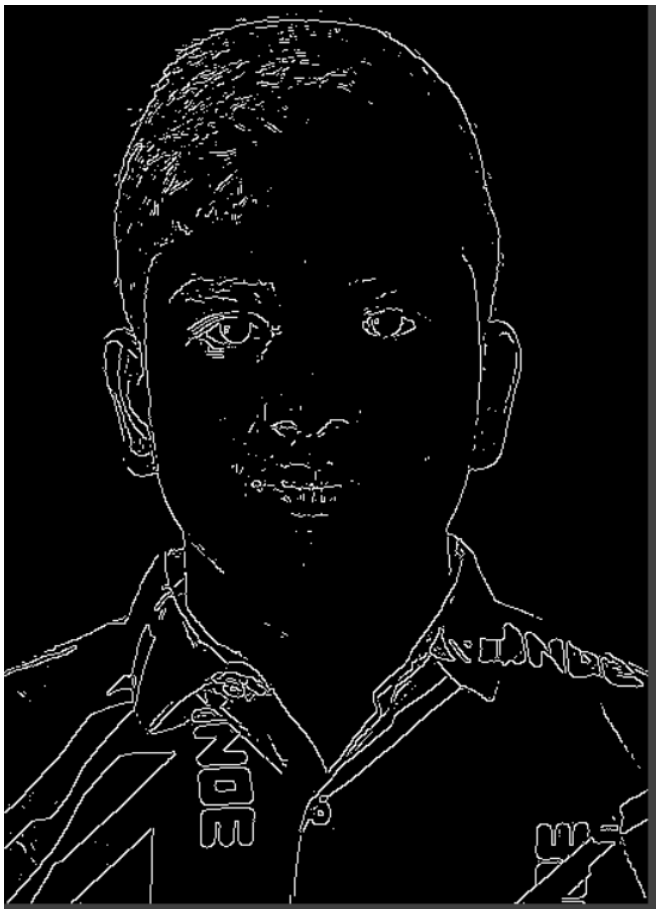
Output-10:



Program-11:

```
edge = cv2.Canny(gray_image,100,100)
cv2.imshow('edge', edge)
```

Output-11:



Program-12:

```
import cv2
import numpy as np
image = cv2.imread('hallphoto3.jpeg')
cv2.imshow('Original Image', image)
cv2.waitKey(0)
Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)
median = cv2.medianBlur(image, 5)
cv2.imshow('Median Blurring', median)
cv2.waitKey(0)
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral Blurring', bilateral)
cv2.waitKey(0) cv2.destroyAllWindows()
```

Output-12:

Original Image:

Gaussian Blur:

Median Blur:

Bilateral Blur: