```
import nltk
import os
import re
import math
import operator
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
nltk.download('averaged_perceptron_tagger')
Stopwords = set(stopwords.words('english'))
wordlemmatizer = WordNetLemmatizer()


def lemmatize_words(words):
    lemmatized_words = []
    for word in words:
        lemmatized_words.append(wordlemmatizer.lemmatize(word))
    return lemmatized_words


def stem_words(words):
    stemmed_words = []
    for word in words:
        stemmed_words.append(stemmer.stem(word))
    return stemmed_words


def remove_special_characters(text):
    regex = r'[^a-zA-Z0-9\s]'
    text = re.sub(regex, '', text)
    return text


def freq(words):
    words = [word.lower() for word in words]
    dict_freq = {}
    words_unique = []
    for word in words:
        if word not in words_unique:
            words_unique.append(word)
    for word in words_unique:
        dict_freq[word] = words.count(word)
    return dict_freq


def pos_tagging(text):
    pos_tag = nltk.pos_tag(text.split())
```

```
    pos_tagged_noun_verb = []
    for word, tag in pos_tag:
        if tag == "NN" or tag == "NNP" or tag == "NNS" or tag == "VB" or
tag == "VBD" or tag == "VBG" or tag == "VBN" or tag == "VBP" or tag ==
"VBZ":
            pos_tagged_noun_verb.append(word)
    return pos_tagged_noun_verb


def tf_score(word, sentence):
    freq_sum = 0
    word_frequency_in_sentence = 0
    len_sentence = len(sentence)
    for word_in_sentence in sentence.split():
        if word == word_in_sentence:
            word_frequency_in_sentence = word_frequency_in_sentence + 1
    tf = word_frequency_in_sentence / len_sentence
    return tf


def idf_score(no_of_sentences, word, sentences):
    no_of_sentence_containing_word = 0
    for sentence in sentences:
        sentence = remove_special_characters(str(sentence))
        sentence = re.sub(r'\d+', '', sentence)
        sentence = sentence.split()
        sentence = [word for word in sentence if word.lower()
                    not in Stopwords and len(word) > 1]
        sentence = [word.lower() for word in sentence]
        sentence = [wordlemmatizer.lemmatize(word) for word in sentence]
        if word in sentence:
            no_of_sentence_containing_word =
no_of_sentence_containing_word + 1
    idf = math.log10(no_of_sentences/no_of_sentence_containing_word)
    return idf


def tf_idf_score(tf, idf):
    return tf*idf


def word_tfidf(dict_freq, word, sentences, sentence):
    word_tfidf = []
    tf = tf_score(word, sentence)
    idf = idf_score(len(sentences), word, sentences)
    tf_idf = tf_idf_score(tf, idf)
    return tf_idf
```

```
def sentence_importance(sentence, dict_freq, sentences):
    sentence_score = 0
    sentence = remove_special_characters(str(sentence))
    sentence = re.sub(r'\d+', '', sentence)
    pos_tagged_sentence = []
    no_of_sentences = len(sentences)
    pos_tagged_sentence = pos_tagging(sentence)
    for word in pos_tagged_sentence:
        if word.lower() not in Stopwords and word not in Stopwords and
len(word) > 1:
            word = word.lower()
            word = wordlemmatizer.lemmatize(word)
            sentence_score = sentence_score + \
                word_tfidf(dict_freq, word, sentences, sentence)
    return sentence_score


file = 'input1.txt'
file = open(file, 'r')
text = file.read()
tokenized_sentence = sent_tokenize(text)
text = remove_special_characters(str(text))
text = re.sub(r'\d+', '', text)
tokenized_words_with_stopwords = word_tokenize(text)
tokenized_words = [
    word for word in tokenized_words_with_stopwords if word not in
Stopwords]
tokenized_words = [word for word in tokenized_words if len(word) > 1]
tokenized_words = [word.lower() for word in tokenized_words]
tokenized_words = lemmatize_words(tokenized_words)
word_freq = freq(tokenized_words)
input_user = int(input('Percentage of information to retain(in
percent):'))
no_of_sentences = int((input_user * len(tokenized_sentence))/100)
print(no_of_sentences)
c = 1
sentence_with_importance = {}
for sent in tokenized_sentence:
    sentenceimp = sentence_importance(sent, word_freq, tokenized_sentence)
    sentence_with_importance[c] = sentenceimp
    c = c+1
sentence_with_importance = sorted(
    sentence_with_importance.items(), key=operator.itemgetter(1),
reverse=True)
cnt = 0
```

```
summary = []
sentence_no = []
for word_prob in sentence_with_importance:
    if cnt < no_of_sentences:
        sentence_no.append(word_prob[0])
        cnt = cnt+1
    else:
        break
sentence_no.sort()
cnt = 1
for sentence in tokenized_sentence:
    if cnt in sentence_no:
        summary.append(sentence)
    cnt = cnt+1
summary = " ".join(summary)
print("\n")
print("Summary:")
print(summary)
outF = open('summary.txt', "w")
outF.write(summary)
```

Output:

```
runfile('C:/Users/Admin/Desktop/swarupa/exp3.py",
wdir='C:/Users/Admin/Desktop/swarupa')
C:\ProgramData\Anaconda3\lib\site-packages\scipy\__init__.py:138:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for
this version of SciPy (detected version 1.23.1)warnings warn (f"A
NumPy version >=(np minversion) and <{np_maxversion} is required for
this version of to
[nltk data] Downloading package averaged perceptron_tagger
[nltk_data] C:\Users\Admin\AppData\Roaming\nltk_data...
Inltk_data] Package averaged perceptron tagger is already up-to
[nltk_data]
date!
return si
Percentage of information to retain(in percent):10
 12
Summary:

SNUG, a joiner. BOTTOM, a weaver. SNOUT, a tinker. Exit PHILOSTRATE
Hippolyta, I woo'd thee with my sword, And won thy love, doing thee
injuries; But I will wed thee in another key, with pomp, with triumph
and with revelling. My noble lord, This man hath my consent to marry
her. Stand forth, Lysander: and my gracious duke, This man hath
bewitch'd the bosom of my child; Thou, thou, Lysander, thou hast
```

given her rhymes, And interchanged love tokens with my child: Thou hast by moonlight at her window sung, with feigning voice verses of feigning love, And stolen the impression of her fantasy with bracelets of thy hair, rings, gawds, conceits, Knacks, trifles, nosegays, sweetmeats, messengers of strong prevailment in unharden'd youth: with cunning hast thou filch'd my daughter's heart, Turn'd her obedience, which is due to me. To stubborn harshness: and, my gracious duke, Be it so she; will not here before your grace Consent to marry with Demetrius, I beg the ancient privilege of Athens, As she is mine, I may dispose of her: Which shall be either to this gentleman or to her death, according to our law Immediately provided in that case. Demetrius, I'll avouch it to his head, Made love to Nedar's daughter, Helena, And won her soul; and she, sweet lady, dotes, Devoutly dotes, dotes in idolatry, Upon this spotted and inconstant man. EGEUS With duty and desire we follow you. to choose love by another's eyes. Sickness is catching: 0, were favour so, Yours would I catch, fair Hermia, ere I go; My ear should catch your voice, my eye your eye, My tongue should catch your tongue's sweet melody. 0, teach me how you look, and with what art You sway the motion of Demetrius' heart. Keep word, Lysander: we must starve our sight From lovers' food till morrow deep