

# Sentiment Analysis of Speech with Application to Various Languages

Akash Apturkar<sup>1</sup>, Alexander I. Iliev<sup>1,2</sup>, Amruth Anand<sup>1</sup>, Arush Oli<sup>1</sup>,  
Swadesh Reddy Siddenki<sup>1</sup>, Vikram Reddy Meka<sup>1</sup>

<sup>1</sup> SRH Berlin University, Charlottenburg, Germany

<sup>2</sup> Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Sofia, Bulgaria  
akash.apturkar1@gmail.com, ailiev@berkeley.edu,  
oliarush1@gmail.com, swadeshsr7@gmail.com,  
amruthanand24@gmail.com, mekavikramreddy@gmail.com

**Abstract.** In this paper we aim to explore and implement a modern speech recognition system using Natural Language Processing (NLP) and sentiment analysis that can be applied in the area of audio and text archive investigation from various languages. To that end we developed a project that can be used to convert speech to text and perform various analysis on a converted text. Furthermore, we focused on recognizing different information such as *names*, *emotions* and also determine the overall *sentiment* of a script. Furthermore, we perform web scraping for names and organizations of importance used in speech. To achieve this, we used Python with various specialized modules. In order to simplify the task of collecting and storing audio inputs for processing we have developed an Android app with connection to a cloud database. This methodology can easily be applied for the purposes of digital presentation and preservation of cultural and scientific heritage.

**Keywords:** Emotion Recognition, Speech Analysis, Language Processing, Android, Digital Archives.

## 1 Introduction

The recent advancement in the field of speech recognition technology has opened the doors to a vast number of use cases for speech based natural language processing (NLP). According to Wikipedia the Google speech recognition API is one of the best speech recognition tool, which can support more than 60 different languages (Google, 2020). The *'SpeechRecognition'* module for Python can be used to easily integrate Google speech recognition API in Python. The module *'textblob'* for NLP can be used to easily translate given text from one language to another. This functionality allows us to perform natural language processing in multiple languages.

Sentiment analysis is a part of NLP that is mainly used for analysis of online reviews or opinion analysis of current topics. For example: sentiment analysis can be used to

perform online opinion analysis of a certain stock on the stock market (Dev Shah, 2018). For this paper we aim to go beyond the general use case of sentiment analysis and explore the possible use of sentiment analysis in the research of preserved literature and audio archives in different languages. Huge archives of radio telecasts, recordings of famous speeches, recitals exist in different countries with different languages. NLP analysis of these archives can help in understand more about the time period and culture. For example: Sentiment analysis of radio broadcasts from world war 2 era from different countries can reveal the general public sentiment at that time and place. Our project can use audio inputs in different languages and can convert them to text using the ‘*SpeechRecognition*’ module in Python, which can be translated for analysis using ‘*textblob*’ library for Python. This allows the project to be used for researching literature and archives in different languages.

The major libraries that can be used for natural language processing and sentimental analysis are ‘*Natural Language ToolKit (NLTK)*’ and ‘*textblob*’ in Python, which we used in our project. For sentiment analysis, NLTK uses the Vader lexicon, which consists of a list of words assign different values of polarity.

## 2 Problem Description and Challenges

To implement our project for use in literature and archives research we had to perform the following tasks:

- Audio input was collected using a microphone or a pre-recorded audio file. The language of the audio had to be detected and converted into a text string using the speech recognition module in Python, which was then converted into English for further analysis using ‘*textblob*’.
- Once we had the text we had to obtain the sentence structure tree by performing *tokenization*, *word lemmatization*, *parts of speech tagging* and *name entity tagging* using NLTK.
- Next we had to extract different emotional states from the text. Various researches have used a number of basic emotions for areas such as: smart ecosystems (Iliev & Stanchev, Smart Ecosystems through Voice and Images, 2020), information retrieval and recommendation (Iliev & Stanchev, Information Retrieval and Recommendation Using Emotion from Speech Signal, 2018), as well as content discovery and perceptual automation (Iliev, Content Discovery Using Perceptual Automation, 2018), where a number of feature vectors have been used as well (Iliev, Feature vectors for emotion recognition in speech, 2016). A monograph on the topic of emotion recognition through speech (Iliev, Emotion Recognition From Speech, 2012) provides a summary of many of the basics in this area. As stated in ‘What is sentiment analysis’, by (Jurafsky, January 27, 2016), the following affective states can be extracted from a given text:
  - a. *Emotion*: brief organically synchronized, evaluation of a major event: *Eg-angry, sad, joyful, fearful, ashamed, proud, elated*

- b. *Mood*: diffuse non-caused low-intensity long-duration change in subjective feeling: Eg- *cheerful, gloomy, irritable, listless, depressed, buoyant*
- c. *Interpersonal stances*: affective stance toward another person in a specific interaction: Eg- *friendly, flirtatious, distant, cold, warm, supportive, contemptuous*
- d. *Attitudes*: enduring, affectively colored beliefs, dispositions towards objects or persons: Eg- *liking, loving, hating, valuing, desiring*
- e. *Personality traits*: stable personality dispositions and typical behavior tendencies: Eg- *nervous, anxious, reckless, morose, hostile, jealous*

For our project we focused on Emotion extraction from the text. We had to calculate the overall sentiment of the text. Then we had to extract the names of people and organizations from the text. Finally, we performed web scraping to evaluate these names with online result analysis.

One of the major challenges in NLP was working with synonyms. For example words like *Saturn, jaguar, or chip* has several different meanings. In different contexts or when used by different people the same term takes on varying referential significance (Hutto & Gilbert, June 2014).

The other major problem in NLP, which specifically affects sentiment analysis, is with detecting *sarcasm*. For example: “*The restaurant was great in that it will make all future meals seem more delicious*” (Farhadloo & Erik, March 2016) is an example of a sarcastic sentence, in which, although there is technically no negative term in the language, it is intended to convey a negative sentiment. This poses a major challenge in assigning negative polarity scores in sentiment analysis.

### 3 System Description

This system is used for analyzing the speech that is provided as an input and to provide the sentiment of the speech is *positive, negative* or *neutral*. In addition, when the system encounters the name in speech production, the web scraping for that name is also carried out. We have also used graphs to show the number of emotions that are present in a speech. In this system the input can be provided through three different modes:

- *Opinion mode*: In opinion mode the input is taken from the web scraping at the respected topic we want to know about. The web scraping is carried out from the online news available on the web and then analysis is carried out on the given topic to see if the topic sentiment is *positive, negative* or *neutral*;
- *Speech mode*: The input in the speech mode is directly taken from the user’s speech in the system. After the speech is provided the system gives us the result for the sentiment analysis for that speech. Furthermore, if the system interacts with the name from the speech then it will carry out the web scraping and the analysis for that name as well;
- *Remote mode*: Remote mode is used to carry out the analysis of the speech that is stored in the bucket. S3 Bucket is a cloud object storage services provided by the Amazon where we store the speech mp3 file that is recorded through our application. For this purpose we have use an AWS (S3) bucket to store the

speech through the application. When the user selects the remote mode then the input is directly fetched from the S3 bucket and the analysis is carried out on the stored speech.

### 3.1 Natural Language ToolKit

Natural Language ToolKit (NLTK) is a python package that works with human language data. It is a free and open source natural language algorithm. There are more than 50 corpora and lexical analysis that provide easy-to-use interfaces like tokenizing, part-of-speech (pos), sentiment analysis, tagging, name entity recognition.

After installing the NLTK package and all requirements were satisfied we used different corpora for analysis (Bird, Loper, & Klein, 2009).

*Tokenize:* Tokenize is a python package, which is used to break the text into simple sentences and the sentences into the word. In our system we used tokenizing for breaking the sentence into words. We can see the code and its output below:

```
#for tokenization
words = word_tokenize(text, "english")
print("Output after tokenization: ")
print(words)

Your input: hello good morning sir this is group 6 and group 7 presenting together and I will be sharing the site and and thank you
Output after tokenization:
['hello', 'good', 'morning', 'sir', 'this', 'is', 'group', '6', 'and', 'group', '7', 'presenting', 'together', 'and', 'I', 'will', 'be', 'sharing', 'the', 'site', 'and', 'and', 'thank', 'you']
```

*Pos tagging:* Pos tagging stands for part of tagging, which is used for defining the properties of the character or words. In other words to define either the word is *noun*, *adverb*, *adjective* we use the pos tagging and its syntax and output is given below:

```
#for Pos
pos = pos_tag(words)
print("\nOutput after parts of speech tagging: ")
print(pos)

Output after parts of speech tagging:
[('hello', 'RB'), ('good', 'JJ'), ('morning', 'NN'), ('sir', 'NN'), ('this', 'DT'), ('is', 'VBZ'), ('group', 'NN'), ('6', 'CD'), ('and', 'CC'), ('group', 'NN'), ('7', 'CD'), ('presenting', 'NN'), ('together', 'RB'), ('and', 'CC'), ('I', 'PRP'), ('will', 'MD'), ('be', 'VB'), ('sharing', 'VBG'), ('the', 'DT'), ('site', 'NN'), ('and', 'CC'), ('and', 'CC'), ('thank', 'VB'), ('you', 'PRP')]
```

*Lemmatization:* Lemmatization is used to discard the unnecessary letters to provide the root words. Likewise, we can see in the output the word ‘comedians’ is changed into ‘comedian’:

```
#for lemmatization
lemmatized_words = []
for word in words:
    word = WordNetLemmatizer().lemmatize(word)
    lemmatized_words.append(word)

Your input: do you think Charlie Chaplin was better than most comedians
Output after tokenization:
['do', 'you', 'think', 'Charlie', 'Chaplin', 'was', 'better', 'than', 'most', 'comedians']
['do', 'you', 'think', 'Charlie', 'Chaplin', 'wa', 'better', 'than', 'most', 'comedian']
```

*Name entities tagging:* The below syntax is used for tagging the name entities. That is when the speech is analysed; the name of a specific person is extracted and then sent for analysis through web scraping:

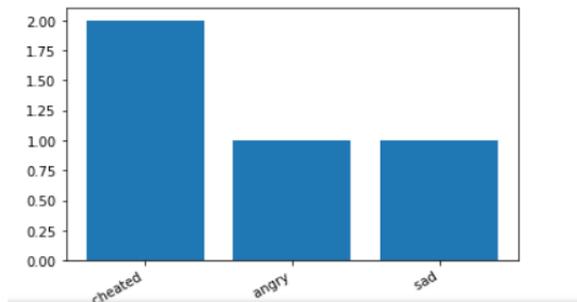
```
#for entities tagging
name_entities = ne_chunk(pos)
print("\nOutput after name entities tagging: ")
print(name_entities)

Output after name entities tagging:
(S
 do/VBP
 you/PRP
 think/VB
 (PERSON Charlie/NNP Chaplin/NNP)
 was/VBD
 better/JJR
 than/IN
 most/JJS
 comedians/NNS)
```

*Emotion recognition:* After all of the tagging and pos are carried out we used the syntax below to calculate the number of emotions that are present in the input. For that, we grouped words with similar emotions, and then we sorted out the list of emotions present in the input as seen in the output below. After the list of emotions was fully fetched we counted the number of repeated emotions in the input:

```
Your input: the accused was angry and felt victimized
Output after tokenization:
['the', 'accused', 'was', 'angry', 'and', 'felt', 'victimized']
List of emotions found in the speech :
['cheated', 'cheated', 'angry', 'sad']

Emotion counter:
Counter({'cheated': 2, 'angry': 1, 'sad': 1})
```



**Fig. 1.** Emotion recognition output with bar graph

### 3.2 VADER Sentiment Analysis

VADER is known as Valence Aware Dictionary and sEntiment Reasoner. It is a lexicon driven and a rule-based sentiment analysis tool. It is an open-source tool under MIT License.

Vader is responsible for generating the *positive*, *negative*, *neutral* and *compound* scores of the given input. Using these scores, the sentiment of a given input can be depicted. Vader not only generates the positive and negative scores but it can also tell us how positive or negative a given input is. Vader Lexicon has more than 9000 features, where each feature has a rating in between -4 (extremely negative) and +4 (extremely positive).

### 3.3 Why VADER?

Though there are different tools for performing the sentiment analysis, we have chosen Vader as it gives the polarity scores mentioned above, which are easy to analyse. Also, Vader performs well in handling the special characters used in a sentence. This tool does not require the training data and it is based on the valence scores in the lexicon.

### 3.4 Vader Sentiment

Vader uses `SentimentIntensityAnalyser()` object, it has a lexicon file with valence ratings and some other set of rules. It uses a `polarity_scores` method to generate the polarity scores:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
sentiment = analyser.polarity_scores('INPUT TEXT')
```

The output from this code will be represented as: `{'neg': 0.0, 'neu': 0.0, 'pos': 0.0, 'compound': 0.0}`

**Compound Score.** Compound score is calculated by the sum of valence scores of the input words and the sum is normalized between -1 (extreme negative) and +1 (extreme positive).

$$\text{compound score} = \frac{\text{sum}}{\sqrt{(\text{sum})^2 + 15}} \quad (1)$$

In the above formula, 15 is the standard value set by Vader to normalize the sum. Compound score of the Vader also depends on many factors like punctuations, idioms, capitalized words etc. Based on the compound score, we can classify the sentences as positive, negative or neutral. Below are the conditions stated to classify:

```
sentiment = analyser.polarity_scores(text)
print(sentiment)
print("\nOverall Sentiment : ")
if sentiment['compound'] < -0.01:
    print("Negative")
elif sentiment['compound'] > 0.05:
    print("Positive")
else:
    print("Neutral")
```

**Calculation of Compound Score.** From the given input sentence, Vader looks for the lexicon words and sum is calculated and normalized. For example, we have an input string as ‘*The king killed his soldier for committing a crime in his kingdom*’.

**Table 1.** Word count example

Sentiment	Words	Count
positive	committing	01
negative	killed, crime	02
neutral	All other words are neutral	09

The input sentence is categorized as one of the above where the sentence is tokenized and the individual words are classified as positive, neutral and negative. Here, only the words that are positive and negative are considered. So, we have *killed* and *crime* as negative and *committing* as a positive word. These three words have a score of -3.5, -2.5 and +0.3 respectively, when summed up we get a total of -5.7. This sum is normalized using the Compound Score formula as below:

```
import math
def normalize(score,alpha=15):
    ss = score/math.sqrt((score*score)+alpha)
    return ss
normalize(-5.7)
Out[34]: -0.8271299960237043
```

From the output, the compound score is: - 0.8271. Comparing this output with Vader analysis output as shown below we notice the same result:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
sentiment = analyser.polarity_scores('the king killed his soldier for
committing a crime in his kingdom')
print(sentiment)
{'neg': 0.437, 'neu': 0.492, 'pos': 0.071, 'compound': -0.8271}

Overall Sentiment :
Negative
```

### 3.5 Speech Recognition

Speech recognition is the subfield of computer science that enables the recognition and translation of human spoken speech into the text format. As our project revolved mainly around sentiment analysis through speech, we used specific Python packages for each part of the speech processing process. Firstly, we used the *speechrecognition* module for recording the speech through the device’s microphone and then we converted it to a text string. Then, we imported the *gtts* (*google-to-text-speech*) package for text-to-speech conversion. Then we used the *playsound* package in order to play the mp3 files.

This function took the input through the microphone and used the google speech recognition system to convert the voice into a text message. Furthermore, we have used *textblob* to detect and translate the text language:

```
Speak now
string in German :
wer sein selbst Meister ist und sich beherrschen kann dem ist die weite Welt und alles untertan
Translated :
whoever is himself a master and can control himself is subject to the wide world and everything
```

## 4 Web Scraping

Once we have extracted the names of people and organizations from speech we run the web scraping script for each of these names. The script can be used to find online results on Google, Google News, Twitter, etc for the names and return a string of text for analysis. This functionality can be useful while researching and analysing archives to extract names of important people and organizations from historical text and evaluate the online opinion for them in news articles, etc.

We used a function for web scraping through the first page of Google news for the 'term' that is passed as a parameter and the result is shown below. There we have used the Python module '*beautifulsoup4*' to traverse HTML output and extract news headlines from the page. Once we received the HTML response, we found all the tags with class name "*BNeawe s3v9rd AP7Wnd*" that held the news headlines. Then we used '*textblob*' to extract only the text from these tags. Next, we detected the language of this extracted text using the method *blob.detect\_language()* from *textblob* and if another language other than English was detected we used the '*blob.translste()*' method to translate it to English. This function returned a string called *out\_str* inside which all the headlines were concatenated. Furthermore, their sentiment analysis can be applied to this *out\_str*. The result below shows a function script & HTML response saved in variable '*soup*':

```
ng-left:16px;}.w1C3Le,.BmP5Tf,.G5Nb8d{padding-left:16px;padding-right:16px;}.G5Nb8d{padding-bottom:12px}
x).G5eFlf{flex:1;display:block}.nMymef span{text-align:center}</style><div><!--SW_C_X--></div><div><div
5cc uUPGi"><div class="kCrYT"><a data-uch="1" href="/url?q=https://screenrant.com/modern-times-charlie-c
-no-cgi/&sa=U&ved=2ahUKEwjM-MSF797pAhUhxYUKHSTxCaAQxfQBMA66BAgBEAE&usg=AOvVaw3r1_iDmC6ZMUKLc
vel="3" class="BNeawe vvwjWb AP7Wnd" role="heading">How Charlie Chaplin Used VFX WAY Before CGI | Screen
="BNeawe UPmit AP7Wnd">Screen Rant</div></a></div><div class="x54gtf"></div><div class="kCrYT"><a data-u
ttps://screenrant.com/modern-times-charlie-chaplin-roller-skating-no-cgi/&sa=U&ved=2ahUKEwjM-MSF
```

The output below shows the production after sentiment analysis when the name Charlie Chaplin is passed as the parameter *term* to the function *web\_scrape()*:

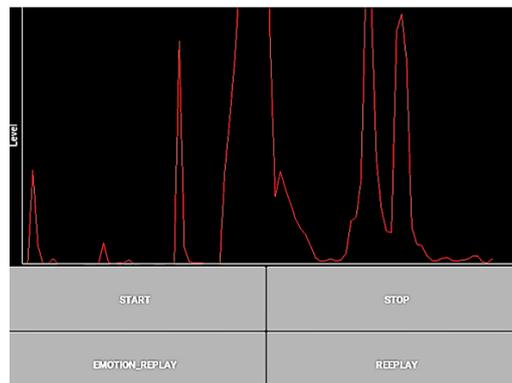
```
Sentiment analysis result for online news about Chaplin,Charlie:
{'neg': 0.02, 'neu': 0.768, 'pos': 0.213, 'compound': 0.9989}

Online opinion of Chaplin,Charlie is
Positive
```

## 5 App Development

As we were in the hunt to create an ecosystem revolving around emotion recognition, there was no better source than a mobile device that could be used to record the input. In the present world the project or concept will connect to the masses quickly with a mobile application, which was the main reason to incorporate this feature. The app has given a whole new dimension for the project as it came with a lot of additional features and advantages. This app has some important features that sets it apart from conventional voice recorders. We built this app on Python 3, using the Kivy library. It was very convenient to deploy the app on Android as Kivy has a dedicated launcher where we could load the project.

*Steps to load the project:* We created a project file by the name Kivy in our android internal storage. In that project file we created a folder by the same name then we uploaded all the files that were used to build the app. The files included *main.py* and all the supporting subordinate files with one text file by the name *android.txt*. This *android.txt* file holds details like Title, Author and Orientation.



**Fig. 2.** This is the GUI of the app

Additional features apart from conventional features are:

- This app consisted of features like: real-time waveforms, recording, playback, delete etc.;
- The app saved the audio files into amazon S3 bucket, which was accessed for further proceedings;
- The app revealed the emotion of the speaker soon after the recording ended;
- The audio was converted into text and was saved in a text file for processing.

Steps for saving the files into S3 bucket on real time:

- We created an AWS account and created a file in the S3 bucket and set it to public;
- We installed AWSCLI using pip and connected awscli with the main account;
- The credentials used to connect to our AWS account were:
  - a. `AWSAccessKeyId - <ENTER AWS S3 ACCESS KEY>`

- b. AWSecretKey - <ENTER AWS S3 SECRET KEY>
- c. Default Regional Name - eu-central-1
- d. S3 bucketname - <ENTER S3 BUCKET NAME>
- e. File name - my.wav

- With these credentials we accessed the file for further processing.

For these processes we used the *boto3* library, which helped us upload/update files to AWS from Python code.

## 6 Model Accuracy Testing

Since used the compound polarity score from *vader.SentimentIntensityAnalyzer()* method from NLTK we wanted to test the model's accuracy in determining *positive*, *negative* and *neutral* sentiments. The ideal scores for determining the sentiment as given in the documentation for Vader (Hutto & Gilbert, June 2014) are as follows:

1. positive sentiment: compound score  $\geq 0.05$
2. neutral sentiment: (compound score  $> -0.05$ ) and (compound score  $< 0.05$ )
3. negative sentiment: compound score  $\leq -0.05$

We tried to determine the ideal values of compound score for positive negative and neutral sentiment by using the following script to pass different values of threshold compound score and check if the model can accurately determine weather the sentences passed are positive or negative. For this we used a set of 5332 positive and negative sentences each (Kinsley, 2020). After installing and importing all dependencies we passed the values in range zero to 0.5 with the step size of 0.1. We passed positive values as threshold to determine positive accuracy and negative values as threshold for negative accuracy like this: first we opened the text document with 5332 positive sentences. For each sentence correctly identified by the model as positive the *pos\_correct* is increased by one and for each sentence can the *pos\_count* is increased by one. To determine the percentage accuracy we divide *pos\_correct* by *pos\_count* and multiply by 100. The same process is followed to calculate negative accuracy. The results for part of the output of accuracy testing are shown below. From them we can see that as we divert from the threshold values zero, while the positive accuracy remains acceptable the negative accuracy decreases rapidly. This is because the model cannot correctly classify *sarcastic* sentences as negative sentences and instead classifies them as *neutral*. Hence, we decided, to set the threshold of negative polarity score to a value closer to zero.

For our project we used these threshold values:

- positive sentiment: compound score  $\geq 0.05$
- neutral sentiment: (compound score  $> -0.01$ ) and (compound score  $< 0.05$ )
- negative sentiment: compound score  $\leq -0.01$

```

Value = 0.01
Positive accuracy = 69.39234808702176% via 5332 samples
Negative accuracy = 40.00375093773443% via 5332 samples
Average = 54.69804951237809

Value = 0.02
Positive accuracy = 69.26106526631658% via 5332 samples
Negative accuracy = 39.90997749437359% via 5332 samples
Average = 54.585521380345085

Value = 0.03
Positive accuracy = 68.82970742685671% via 5332 samples
Negative accuracy = 39.36609152288072% via 5332 samples
Average = 54.09789947486872

Value = 0.04
Positive accuracy = 68.77344336084022% via 5332 samples
Negative accuracy = 39.19729932483121% via 5332 samples
Average = 53.98537134283571

Value = 0.05
Positive accuracy = 68.67966991747937% via 5332 samples
Negative accuracy = 39.122280570142536% via 5332 samples
Average = 53.900975243810954

Value = 0.06
Positive accuracy = 67.98574643660915% via 5332 samples
Negative accuracy = 38.40960240060015% via 5332 samples
Average = 53.19767441860465

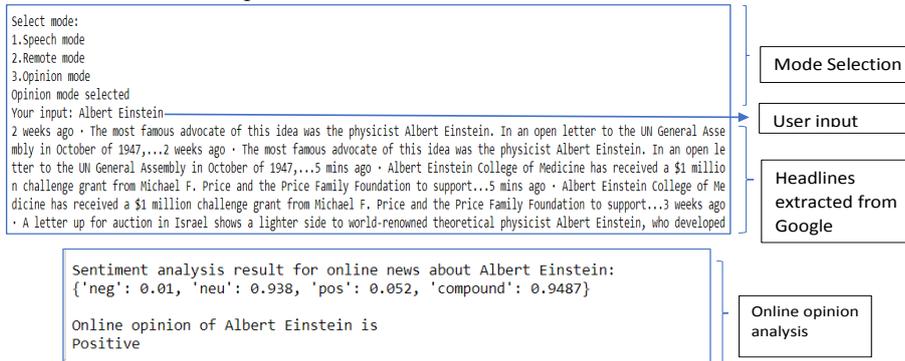
```

## 7 Results and Discussion

The main objective of our research project was to build Smart Speech Ecosystem, which mainly uses the Natural Language Tool Kit (NLTK). The key result obtained from this research is ideal value for the compound polarity.

- Positive Sentiment: compound score  $\geq 0.5$
- Neutral Sentiment: (compound score  $\geq -0.01$ ) and (compound score  $< 0.05$ )
- Negative Sentiment: compound score  $\leq -0.01$

A. *Opinion Mode*: In opinion mode, the input we had taken from the user performed web scraping using beautiful soup method that extracts headlines from Google and here is the output:



B. *Speech Mode*: In speech mode we took the input from the user and processed through Google-Text-To-Speech (GTTS) it converted the user speech into text and performs sentiment analysis as follows:

- *word tokenization* breaks a piece of text into words:

```
Speech mode selected
Your input: among the festive atmosphere some politicians accused Prime Minister Winston Churchill of having cheated the Indian people and caused the famine
Output after tokenization:
['among', 'the', 'festive', 'atmosphere', 'some', 'politicians', 'accused', 'Prime', 'Minister', 'Winston', 'Churchill', 'of', 'having', 'cheated', 'the', 'Indian', 'people', 'and', 'caused', 'the', 'famine']
```

- then we used as a function to perform parts of *speech tagging (PoS)* input and assigns a certain tag to the word list:

```
Output after parts of speech tagging:
[['among', 'IN'], ['the', 'DT'], ['festive', 'NN'], ['atmosphere', 'RB'], ['some', 'DT'], ['politicians', 'NNS'], ['accused', 'VBN'], ['Prime', 'NNP'], ['Minister', 'NNP'], ['Winston', 'NNP'], ['Churchill', 'NNP'], ['of', 'IN'], ['having', 'VBG'], ['cheated', 'VBN'], ['the', 'DT'], ['Indian', 'JJ'], ['people', 'NNS'], ['and', 'CC'], ['caused', 'VBD'], ['the', 'DT'], ['famine', 'NN']]]
```

- the output of PoS tagging was passed as an input to the *word lemmatization*, which splits the text into individual words based on their root words;
- in name entity tagging the input is taken from word lemmatization, from *name parser* library (Gulbranson, 2018) which we are importing. The *HumanName().last* and *HumanName().first* methods were used to find the last and first names, also forming an entity tree:

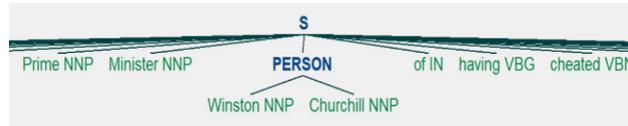
```
Sentiment analysis result for online news about Churchill,Winston:
{'neg': 0.087, 'neu': 0.886, 'pos': 0.027, 'compound': -0.9861}

Online opinion of Churchill,Winston is
Negative
```

Then we represented the *Name entities tagging* like this:

```
Output after name entities tagging:
(S
  among/IN
  the/DT
  festive/NN
  atmosphere/RB
  some/DT
  politicians/NNS
  accused/VBN
  Prime/NNP
  Minister/NNP
  (PERSON Winston/NNP Churchill/NNP)
  of/IN
  having/VBG
  cheated/VBN
  the/DT
  (GPE Indian/JJ)
  people/NNS
  and/CC
  caused/VBD
  the/DT
  famine/NN)
```

- from the *Human Name library*, a name parser method was used to find the last and first names then an entity tree was formed. Figure 3 below shows the sentence structure tree, which is cropped for better view:



**Fig. 3.** This is the sentence structure tree (cropped)

- we then found the list of emotions from the speech and represented through data visualization;
- we performed the overall sentiment analysis to the above speech and we got the *positive*, *negative*, *neutral* and *compound* scores for the speech:

```

Sentiment analysis result:
{'neg': 0.208, 'neu': 0.679, 'pos': 0.113, 'compound': -0.3612}

Overall Sentiment :
Negative

Name entities detected in the speech are :
Churchill,Winston
Running Online opinion analysis of Churchill,Winston
  
```

- based the results of sentiment analysis we conducted online news search on the particular person.

C. *Remote Mode*: In this research paper we had one more interesting feature of Android connect application, which was developed using the Kivy library. The input from the app was taken and connected to Amazon Web Service (AWS). The data was stored in a bucket through *boto3* method and connected to our training model that performed online analysis and calculated the compound score accordingly.

## 8 Conclusion and Future Work

In this research, sentiment analysis of text through speech recognition was tested using the natural language toolkit (NLTK) and Kivy. We firstly took the speech input and then converted it to text through GTTS. Then we performed the sentiment analysis using word tokenisation, parts of speech, lemmatization, entity tagging and also, we used Human Name library for identifying first names and last names. In the next portion of our work, we performed web scraping for deeper analysis of the text. Finally, online opinion analysis was conducted through sentiment intensity analyser to get their positive, negative, neutral scores of the text in order to calculate compound polarity. Furthermore, we had displayed the data visualization chart how the emotion was carried out. One of the most important things we developed was connecting remote mode to our training model. We used amazon web services as a remote mode, connected by importing *boto3* to the target model. The NLTK was used to achieve this sentiment analysis. This system can be successfully embedded in areas such as discovery and preservation of cultural and scientific heritage as well any kind of investigative and analytical work that deals with recorded speech and text.

To take a step further, we are planning to import data from different languages as it comes available to us. We are also focusing on historical data, ancient books in particular in order to perform deeper analysis of the content and specific historical facts. In addition, we are planning to develop multimedia attributes like video capturing of the historical data and photos of the ancient figures (kings, queens, etc.) based on the requirements we set for the project.

## Acknowledgments

This work was partially supported by the Bulgarian Ministry of Education and Science under National Scientific Program “Information and Communication Technologies for a Single Digital Market in Science, Education and Security”, approved by DCM No 577, 17 August 2018.

## References

- Google. (05 March 2020). *Cloud Speech-to-Text API - Release notes*. Accessed at: <https://cloud.google.com/speech-to-text/docs/release-notes>.
- Dev Shah, H. I. (December 2018). *Predicting the Effects of News Sentiments on the Stock Market*. Свалено от arXiv.org
- Iliev, A. I., & Stanchev, P. L. (2020). Smart Ecosystems through Voice and Images. *Proceedings of 35th International Conference on Computers and Their Applications, CATA 2020*. 69, pp. 256-263. San Francisco, CA, USA: EPiC Series in Computing.
- Iliev, A. I., & Stanchev, P. L. (2018). Information Retrieval and Recommendation Using Emotion from Speech Signal. *IEEE Conference on Multimedia Information Processing and Retrieval* (pp. 222-225). Miami, FL, USA: IEEE.
- Iliev, A. I. (2018). Content Discovery Using Perceptual Automation. *Proceedings of the 10th International Conference on Management of Digital EcoSystems (MEDES'18)* (pp. 233-238). Tokyo, Japan: ACM, New York, NY-USA.
- Iliev, A. I. (2016). Feature vectors for emotion recognition in speech. *National Informatics Conference* (pp. 225-238). Sofia, Bulgaria: Bulgarian Academy of Sciences, Mathematics and Informatics Department.
- Iliev, A. I. (2012). *Emotion Recognition From Speech* (Vol. 1). Lambert Academic Publishing.
- Jurafsky, D. (January 27, 2016). *What is sentiment analysis*. Stanford University. Accessed at: <https://web.stanford.edu/class/cs124/lec/sentiment.pdf>.
- Hutto, C., & Gilbert, E. E. (June 2014). VADER: A Parsimoni on Weblogs and Social Media. *VADER: A Parsimoni on Weblogs and Social Media*. Ann Harbor, MI, USA: ICWSM-14. Accessed at: <https://pythonprogramming.net/sentiment-analysis-python-textblob-vader/>.
- Farhadloo, M., & Erik, R. (March 2016). Fundamentals of Sentiment Analysis and Its Applications. In W. Pedrycz, & S.-M. Chen (Eds.), *Sentiment Analysis and Ontology Engineering*. Springer International Publishing.

- Bird, S., Loper, E., & Klein, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.
- Kinsley, H. (n.d.). *Out of the Box Sentiment Analysis options with Python using VADER Sentiment and TextBlob*. Retrieved June 10, 2020, Accessed at: [PythonProgramming.net](http://PythonProgramming.net).
- Gulbranson, D. (2018). *Nameparser Documentation Release 1.0.2*. Retrieved June 10, 2020, Accessed at: [readthedocs.org](http://readthedocs.org).

Received: June 10, 2020  
Reviewed: June 25, 2020  
Finally Accepted: June 30, 2020

