I decided to use Python for programming this assignment. Python has made it very easy for natural language processing by importing NLTK which is the Natural Language Toolkit. I looked at the assignment broadly and categorized the sentiments into positive, negative and neutral.

I first converted speech to text using the google speech to text converter api call.

I tokenized the sentence into words and then converted them into root form by stemming. This step is essential because words may appear in various forms like past tense, present participle etc and we have to make sure that the words are consistent while comparing.

I created pre-defined lists consisting of words relevant to positive, negative and neutral sentiments from fig 6 of the https://www2.bc.edu/~russeljm/publications/Russell1980.pdf reading mentioned in the assignment.

I converted the 3 lists into root forms for consistent comparison.

For each word in the text, I maintained the counts of the number of words that are either positive, negative or neutral by creating a hash table or a dictionary.

I compared the counts of each of the sentiments and printed out the sentiment for the max count. I also cumulatively added the counts for each sentiment for each audio file and found the max count for the aggregated sentiment.

Points of improvement in the code:

1) I ran into multiple infrastructure issues which took away a lot of time. I was trying to look for any documentation which could help me import the ANEW dictionary to calculate sentiments based on the valence as mentioned in the paper, but I couldn't find any due to the time constraint. This could have brought in more sophistication in my code. For example we can import Wordnet dictionary in python NLTK and fetch the synonyms, antonyms etc for a particular word. Hence I had to construct pre-defined lists of words for positive, negative and neutral sentiments which are very small.
2) If I was given training corpus of various sentiments, I could have built a deep dictionary of my own. I could have achieved sentiment classification on the audio file text by first building a dictionary of unique words from the training data. I then could construct a 2D bigram table with the unique words. This would include both the seen and the unseen bigrams from the training corpus. I could record the frequencies of the bigrams. I could smooth this frequency with Good Turing smoothing and assign a non-zero probability to each of the bigrams in the 2D array. I could also handle unknown words by replacing words with frequency=1 by "UNK". This will help assign non-zero probabilities to the words that are present in the audio file text corpus but not in the training corpus.
   Once the above pre-processing is done on the training corpus, I could then look at the bigrams in the audio file text corpus and look up the previously created dictionary from the training corpus for the probabilities. If there is no bigram present, the value of "UNK" would be assigned to that particular bigram. Using the formula for calculating perplexity, we would end up with values for each of the training corpus sentiment classification. The lower the perplexity, the higher is the inclination towards the classification of the training data. The lowest value would indicate the sentiment classification of the audio file text data.