Nile University

Spring 2019

SPEECH-BASED CHATBOT EXTRACTING USER'S INTERESTS

A Project By: Aya Abo Hagag & Shereen Alaa

ECEN-550: Natural Language Processing

DR. NERMIN NEGEID

Table of Contents

1. **Introduction**

Chatbots represent a potential shift in how people interact with data and services online. Chatbots are machine agents that serve as natural language user interfaces for data and service providers. Currently, chatbots are typically designed and developed for mobile messaging applications. The current interest in chatbots is spurred by recent developments in artificial intelligence (AI) and machine learning. Major Internet companies such as Google, Facebook, and Microsoft see chatbots as the next popular technology. In Spring 2016, Facebook and Microsoft provided resources for creating chatbots to be integrated into their respective messaging platforms, Messenger and Skype. One year later, more than 30,000 chatbots have been launched on Facebook Messenger. Other messaging platforms have also seen a substantial increase in chatbots, including Slack, Kik, and Viber.

Chatbots are seen as means for direct user or customer engagement through text messaging for customer service or marketing purposes, bypassing the need for special-purpose apps or webpages. A huge shift also is developing voice chatbots, which can a understand a talking user and respond to him in real-time. The direction followed for such chatbots is using voice queries to answer questions, perform actions and make recommendations according to the user's needs. This can be seen in Siri which is considered as an intelligent assistant which enables Apple device users to command actions or queries from the device. For this project, a voice chatbot is developed but not similar to the usual voice chatbots; the suggested chatbot will understand a talking user as expected but it will not be directed for answering user queries. It will be more of a multi-purpose speech based chatbot that can display a range of news feeds or something like google cards related to his speech. The news feeds can be commercial and include advertisements or simply articles.

2. **Project Management Plan**

In this segment of the documentation, we describe the project as an overview of scope and objectives.

2.1. **Project Scope**

The idea of this project revolves around the obtaining a user's interest via speech; the method of obtaining speech can be through voice messages or calls depending on the platform. However, for security and integrity purposes, in this project, we utilize recorded messages.
This generic project can be used for multiple purposes, including, personalized advertisements, personalized newsfeed, etc. all based on just knowing a user's interest.

2.2. <u>Project Objectives</u>

Regarding functional requirements, the project should be able to:

- Listen and record a user’s input
- Convert speech to text
- Preprocess text and refine it using natural language processing techniques
- Input prepared text to a classifier
- Output user interest according to categories

As for non-functional requirements:

- Performance; being that the product is made for multiple purposes, it should be light, easy to use, and quick for a user.
- Reliability; it should somehow serve in a high accuracy rate.
- Availability; to apply the first two requirements in an orderly manner, there must be a stable internet connection.

3. <u>Analysis</u>

In this segment of the documentation, we introduce an outlined research based on our suggested model, which includes comparisons as to why we might favor a tool over another. This segment also includes our choice of tools and the reason we did so.

Recall the project objectives as follows,

A. The chatbot will listen to the user and convert its speech to text.

B. The text will be processed using some natural processing techniques such as tokenization and part of speech tagging. The purpose of preprocessing is to obtain the valuable words that can be used for finding the user interests.

C. A classifier is then used for the final stage; the output text or bag of words from step two is classified based on Naïve Bayes classifier to produce a list of interests for the user.

For every step mentioned above, there was a lot of options to choose from so we researched the most applicable and suitable option to use.

3.1. Speech to Text

To convert speech to text in python, there are a lot of engines and APIs that can do this job. A library for performing speech recognition named PyPI was found.

Speech recognition engine/API support:

- CMU Sphinx (works offline)
- Google Speech Recognition

- Google Cloud Speech API
- Wit.ai
- Microsoft Bing Voice Recognition
- Houndify API
- IBM Speech to Text
- Snowboy Hotword Detection (works offline)

Initially, we were inclined towards using the CMU Sphinx being that it works offline, but on testing, we found that it was not at all accurate. Therefore, we preferred the use of Google speech recognition API as it converts with great accuracy in a small response time.

3.2. Preprocessing

During this phase, we create a table of comparison for the important points to be considered in choice between IBM Watson, Stanford NLP, and NLTK in terms of natural language processing techniques.

IBM Watson	Stanford NLP	NLTK
Analyze semantic features of text input, including categories, concepts, emotions, entities, keywords, metadata, relations, semantic roles, and sentiment.	Part-of-speech (POS) tagger, Named Entity Recognizer (NER), Co-reference resolution system, sentiment analysis, bootstrapped pattern learning, and open information and extraction tools.	A suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.
Supports URL, Text, HTML as input types	Supports only Text as input type	Supports Text
Support Arabic, English (only English supports all features)	Support Arabic, English (only English supports all features)	Support Arabic, English (only English supports all features)
Pricing ● Lite- free ● Standard- paid (based on features and services selected)	Pricing Not free and open source	Pricing Free and open source

Supports python	Supports python	Supports python

From this comparison, we can see that the three tools can support text as input type, python as language programming, English but only one of them is totally free which is NLTK making it the favorable choice. Another major difference is what they can do in terms of natural language processing; for our chatbot, NLTK can very sufficient although we might use Stanford NLP as it’s more powerful in Named Entity recognition.

According to another resource, we found that comparing accuracy between Stanford and NLTK with regards to named entity recognition, Stanford surpasses NLTK by 2.52%, which supports the second non-functional requirement mentioned earlier.

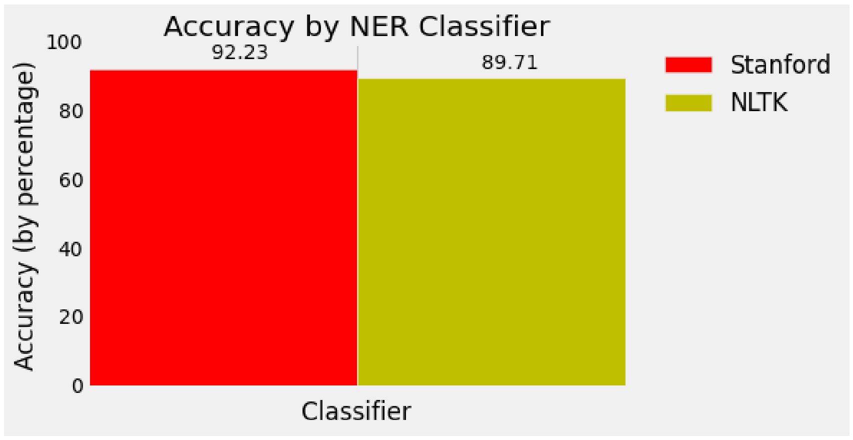


Figure 1: Stanford vs. NLTK NER Accuracy

Now comparing speed of the same aspect, Stanford -once again- surpasses NLTK by a 7.72 ratio as shown below.

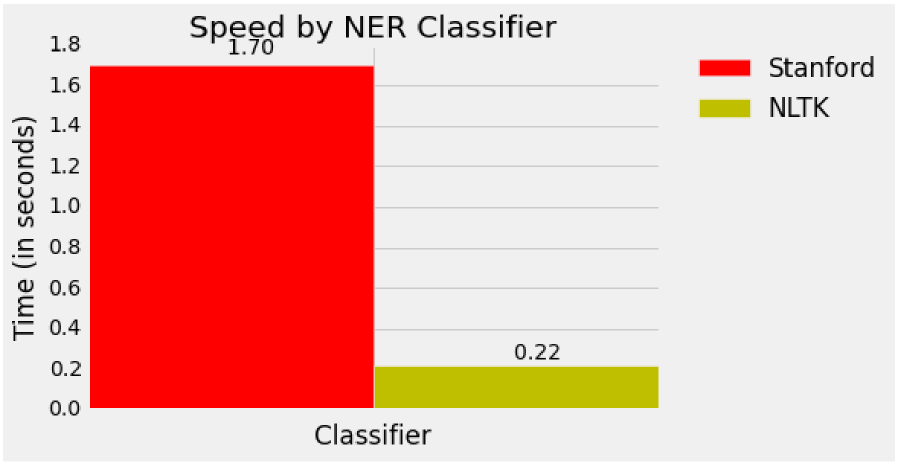


Figure 2: Stanford vs. NLTK NER Speed

Therefore, based on the aforementioned graphs, we decided to use Stanford for named entity recognition.

3.3. Classification

For this phase, a classifier named Multinomial Naive Bayes will be used. The reason behind choosing this classifier is that it's surprisingly effective and commonly used as it's a classic algorithm in text classification.

This classifier is "naive" because it assumes independence between "features", in this case: words. Said differently: each word is treated as having no connection with other words in the sentence being classified.

4. <u>Design</u>

The model design can be illustrated in the diagram below as a systematic approach, in which we try our best to encompass and apply the main concepts from the course content.

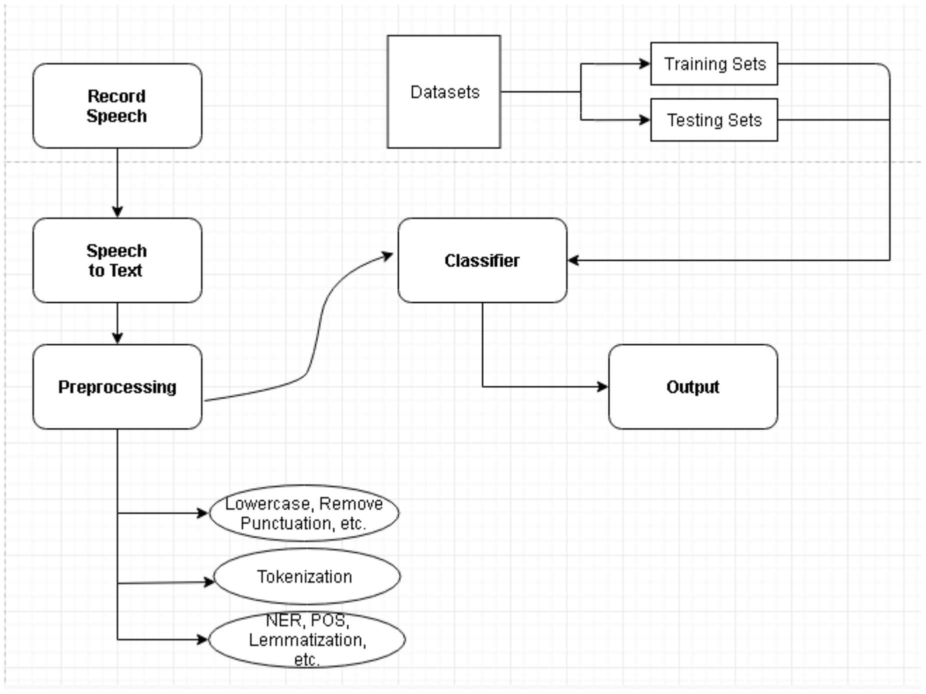


Figure 3: System Model

The preprocessing segment includes the following:

1. Convert text to lowercase; a pure python implementation.
2. Remove punctuation; utilizing regular expressions.
3. Tokenize text; NLTK implementation.
4. Remove stop words; NLTK implementation.

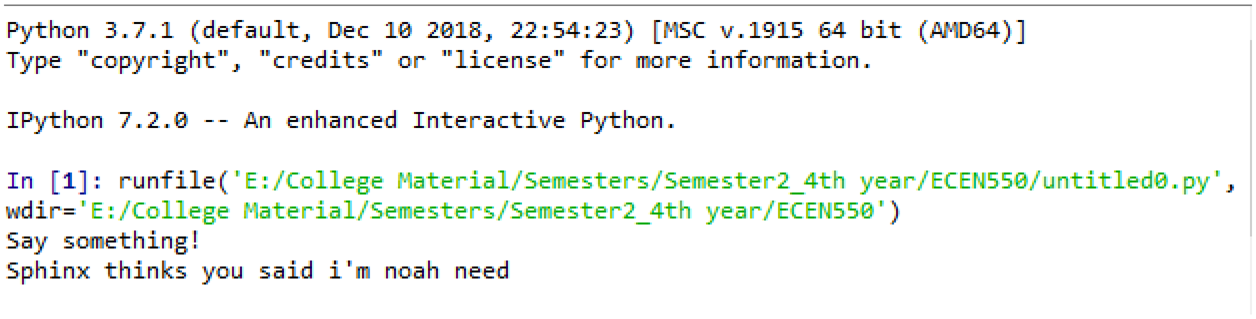
5. Lemmatize words to base forms; during design, we were only considering using NLTK, but later found out there are other libraries to do so.
6. Add part of speech tagging; NLTK.
7. Resources mentioned chunking; which is basically shallow parsing or grouping based on regular expressions chunk grammar—can be POS.
8. Identify named entity recognition; design-wise, NLTK.
9. Feed the output of the previous cumulative steps to the classifier, which must be trained and tested by segregated datasets.

5. <u>Implementation</u>

In this fragment of the documentation, we include code snippets, constraints in implementation, etc. to efficiently highlight the most imperative aspects of the project.

5.1. Speech to Text

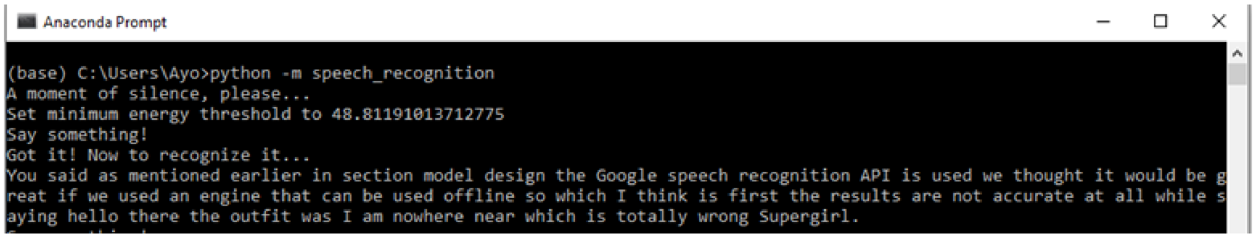
As mentioned earlier in section model design, the google speech recognition API is used. We thought it would be great if we used an engine that can be used offline, so we tried sphinx first; the results were not accurate at all. While saying “hello there”, the output was “I’m noah need” which is totally wrong. See figure below.



```
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/untitled0.py',
wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
Say something!
Sphinx thinks you said i'm noah need
```

Figure 4: Implementing Speech Recognition via Sphinx

We then used Google speech recognition and the results were great with high accuracy in real time. See snippet below.

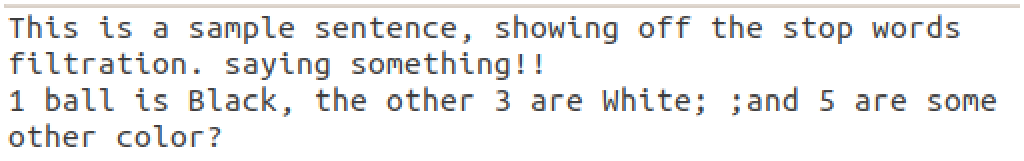


```
(base) C:\Users\Ayo>python -m speech_recognition
A moment of silence, please...
Set minimum energy threshold to 48.81191013712775
Say something!
Got it! Now to recognize it...
You said as mentioned earlier in section model design the Google speech recognition API is used we thought it would be g
reat if we used an engine that can be used offline so which I think is first the results are not accurate at all while s
aying hello there the outfit was I am nowhere near which is totally wrong Supergirl.
```

Figure 5: Implementing Speech Recognition via Google

5.2. Preprocessing

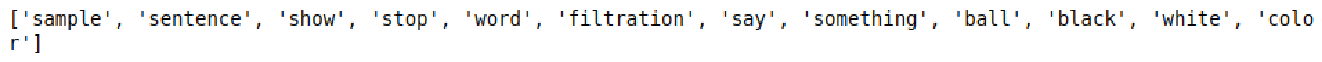
According to a source, pre-processing is required to “clean” up the text required to process whether or not machine learning is applied to enhance functionality and efficiency of results.
Therefore, in this project, we eliminate dispensable data from the input text, which includes, numbers, punctuation, and stop words, which can be considered the cleaning segment of preprocessing. Next, tokenization is applied, firstly sentence and then word to separate to apply future changes to these respective tokens. Then we apply part of speech tagging, which can be considered as the annotation of preprocessing, to the word tokens to be able to use it in lemmatization, or in other words, the normalization of text.
additionally, we apply named entity recognition, which was Stanford-based, to classify word tokens in terms of person, location, etc. Although NER is not used for a specific purpose in this version of the project, it will be used in updated versions.
Firstly, we test with hand-written text to ensure pre-processing works exactly as required with the following text file,



```
This is a sample sentence, showing off the stop words
filtration. saying something!!
1 ball is Black, the other 3 are White; ;and 5 are some
other color?
```

Figure 6: Snippet of example text

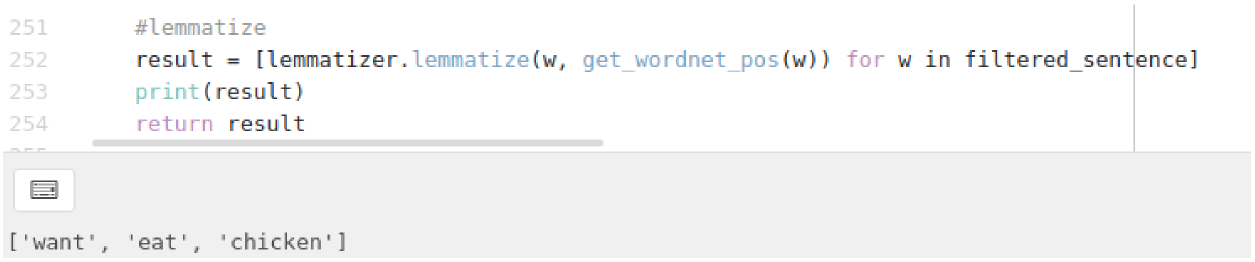
which yields the following result,



```
['sample', 'sentence', 'show', 'stop', 'word', 'filtration', 'say', 'something', 'ball', 'black', 'white', 'colo
r']
```

Figure 7: Result of example text

Next, we test on actual recorded speech, so saying “I want to eat chicken”, yields as follows,



```
    #lemmatize
    result = [lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in filtered_sentence]
    print(result)
    return result
```

```
['want', 'eat', 'chicken']
```

Figure 8: Result of recorded voice

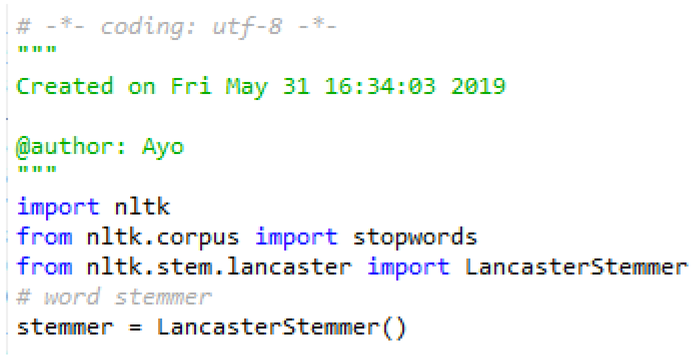
The result of these processes is then fed to the classifier, which in turn returns the interest of the user's speech.

5.3. Classification

The classification process is based on the text classifier, Naive Bayes. Certain steps are followed to implement the classifier.

1. Refer to libraries we need
2. Provide training data
3. Organize our data
4. Iterate: code + test the algorithms needed
5. Abstract algorithm

Step 1(importing the needed libraries): We begin by importing our natural language toolkit.



```
# -*- coding: utf-8 -*-
"""
Created on Fri May 31 16:34:03 2019

@author: Ayo
"""
import nltk
from nltk.corpus import stopwords
from nltk.stem.lancaster import LancasterStemmer
# word stemmer
stemmer = LancasterStemmer()
```

Figure 9: Importing NLTK in Spyder editor supporting python

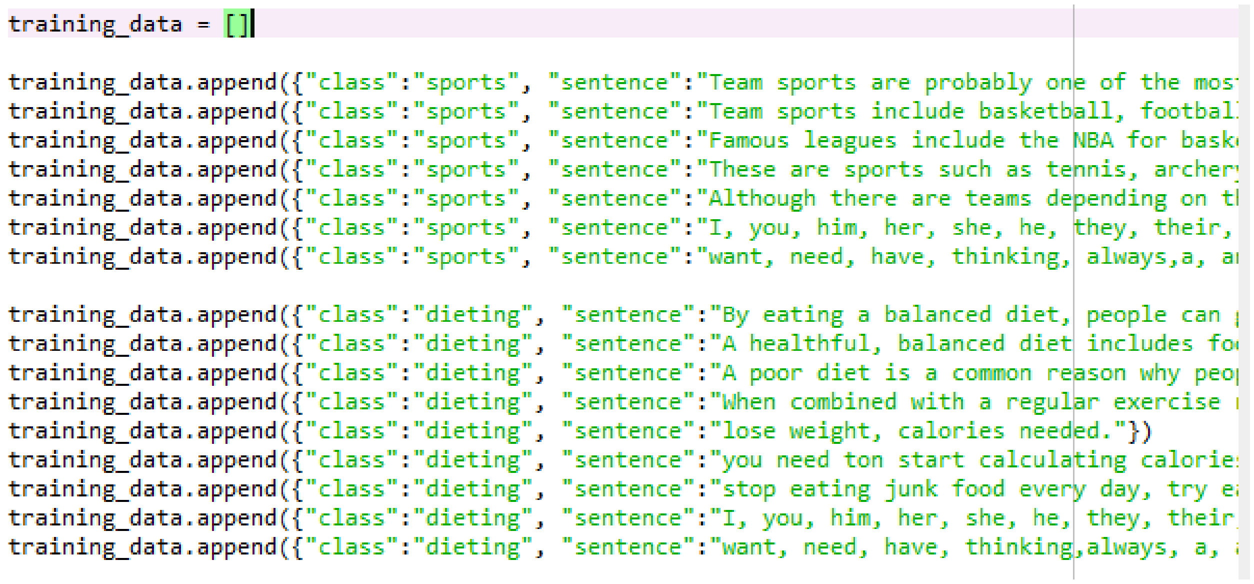
We use NLTK (natural language toolkit) for 2 things:

- Breaking up sentences into words (**tokenization**): "I enjoy cooking" tokenizes to a list of individual words: "I", "enjoy", "cooking".
- Reducing words to their stem (**stemming**): "have" stems to "hav" which allows it to be matched with "having" (same stem)

There are various stemmers that we can use, for our project we'll use the Lancaster stemmer. Next step is to provide some training data. Several sentences that are associated with each class. If the user says, "I enjoy cooking", we want that to be the "cooking" class.

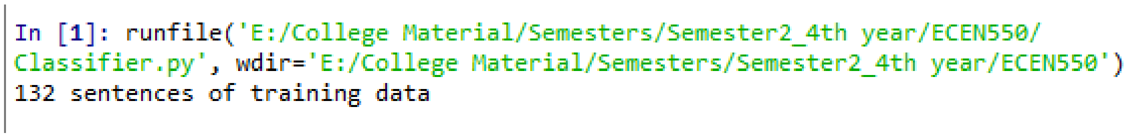
Step 2 (creating the dataset): For the purpose of this project, we have created our own dataset to be used as training data. The dataset is not big for now, but we intend to make it cover more interests. The dataset covers 11 interests: sports, dieting, programming, Natural Language Processing, online courses, psychology, philosophy, drawing and painting, playing music, shopping and online shopping, perfumes, and cooking. The Dataset created will be showed in detail in section Test Documentation.

Here a snippet of the dataset represented in the code and the number of sentences in the dataset.



```
training_data = []

training_data.append({"class":"sports", "sentence":"Team sports are probably one of the mos
training_data.append({"class":"sports", "sentence":"Team sports include basketball, footbal
training_data.append({"class":"sports", "sentence":"Famous leagues include the NBA for bask
training_data.append({"class":"sports", "sentence":"These are sports such as tennis, archer
training_data.append({"class":"sports", "sentence":"Although there are teams depending on t
training_data.append({"class":"sports", "sentence":"I, you, him, her, she, he, they, their,
training_data.append({"class":"sports", "sentence":"want, need, have, thinking, always,a, a

training_data.append({"class":"dieting", "sentence":"By eating a balanced diet, people can
training_data.append({"class":"dieting", "sentence":"A healthful, balanced diet includes fo
training_data.append({"class":"dieting", "sentence":"A poor diet is a common reason why peo
training_data.append({"class":"dieting", "sentence":"When combined with a regular exercise
training_data.append({"class":"dieting", "sentence":"lose weight, calories needed."})
training_data.append({"class":"dieting", "sentence":"you need ton start calculating calorie
training_data.append({"class":"dieting", "sentence":"stop eating junk food every day, try e
training_data.append({"class":"dieting", "sentence":"I, you, him, her, she, he, they, their
training_data.append({"class":"dieting", "sentence":"want, need, have, thinking,always, a,
```

Figure 10: Dataset

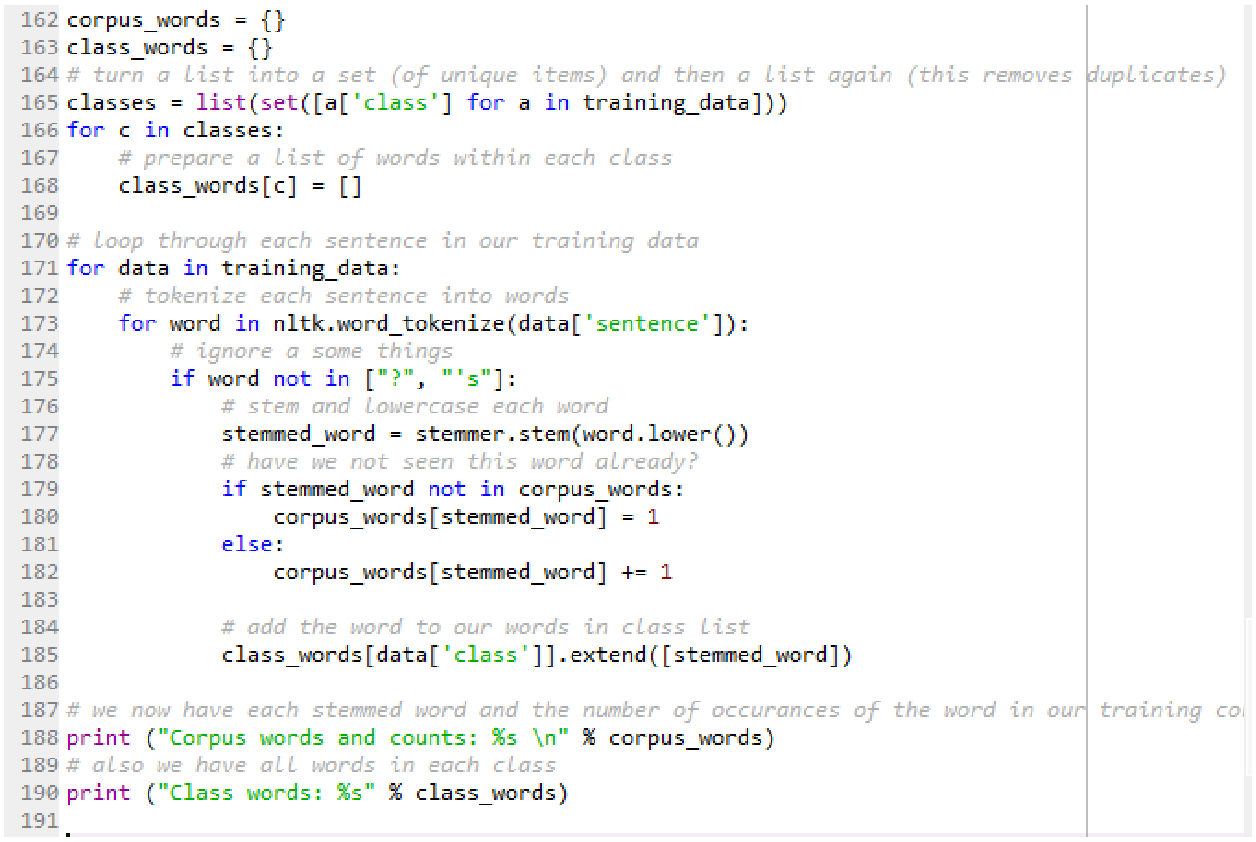


```
In [1]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/
Classifier.py', wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
132 sentences of training data
```

Figure 11: Output snippet showing total number of datasets

Notice that this is a list [] or dictionaries {}. Each dictionary has attributes: “class” and “sentence”. Other attributes could easily be added. This is considered to be a minimal data to show the inner workings of this algorithm, in producing a full application of the chat-bot, it might have hundreds of classes, each with several training sentences.

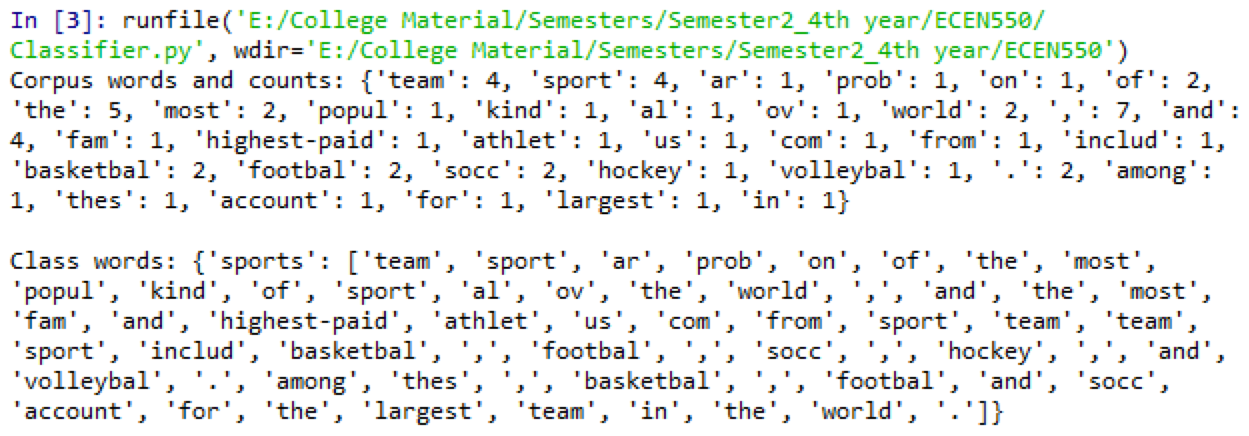
Step 3 (organizing the data): The next step is to organize our data in structures that can be worked algorithmically.



```python
corpus_words = {}
class_words = {}
# turn a list into a set (of unique items) and then a list again (this removes duplicates)
classes = list(set([a['class'] for a in training_data]))
for c in classes:
    # prepare a list of words within each class
    class_words[c] = []

# loop through each sentence in our training data
for data in training_data:
    # tokenize each sentence into words
    for word in nltk.word_tokenize(data['sentence']):
        # ignore a some things
        if word not in ["?", "'s"]:
            # stem and lowercase each word
            stemmed_word = stemmer.stem(word.lower())
            # have we not seen this word already?
            if stemmed_word not in corpus_words:
                corpus_words[stemmed_word] = 1
            else:
                corpus_words[stemmed_word] += 1

            # add the word to our words in class list
            class_words[data['class']].extend([stemmed_word])

# we now have each stemmed word and the number of occurances of the word in our training co
print ("Corpus words and counts: %s \n" % corpus_words)
# also we have all words in each class
print ("Class words: %s" % class_words)
```

Figure 12: Handling training sets

Review the output relative to our training data. The output includes only the first 2 sentences of class "sports". The rest of training data are commented for simplicity while running and for simplicity of the output.

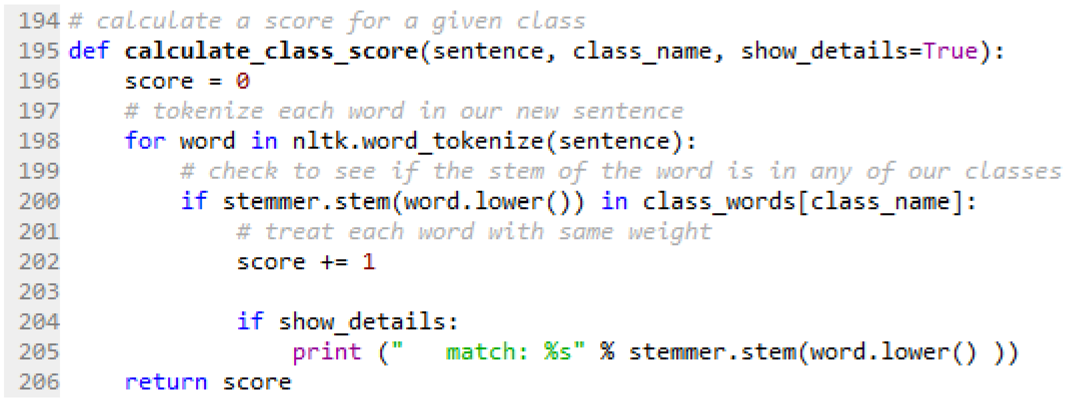


```
In [3]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/
Classifier.py', wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
Corpus words and counts: {'team': 4, 'sport': 4, 'ar': 1, 'prob': 1, 'on': 1, 'of': 2,
'the': 5, 'most': 2, 'popul': 1, 'kind': 1, 'al': 1, 'ov': 1, 'world': 2, ',': 7, 'and':
4, 'fam': 1, 'highest-paid': 1, 'athlet': 1, 'us': 1, 'com': 1, 'from': 1, 'includ': 1,
'basketbal': 2, 'footbal': 2, 'socc': 2, 'hockey': 1, 'volleybal': 1, '.': 2, 'among':
1, 'thes': 1, 'account': 1, 'for': 1, 'largest': 1, 'in': 1}

Class words: {'sports': ['team', 'sport', 'ar', 'prob', 'on', 'of', 'the', 'most',
'popul', 'kind', 'of', 'sport', 'al', 'ov', 'the', 'world', ',', 'and', 'the', 'most',
'fam', 'and', 'highest-paid', 'athlet', 'us', 'com', 'from', 'sport', 'team', 'team',
'sport', 'includ', 'basketbal', ',', 'footbal', ',', 'socc', ',', 'hockey', ',', 'and',
'volleybal', '.', 'among', 'thes', ',', 'basketbal', ',', 'footbal', 'and', 'socc',
'account', 'for', 'the', 'largest', 'team', 'in', 'the', 'world', '.']}
```

Figure 13: Handling corpus

Notice the "corpus" (an NLP term) a collection of all stemmed words. The stem of "include" is "includ" so that it matches the stems for "including", as a simple example. We've now organized our data into 2 dictionaries: corpus_words (each stemmed word and the # of occurrences) class_words (each class and the list of stemmed words within it). Our algorithm will use these data structures to do its thing.

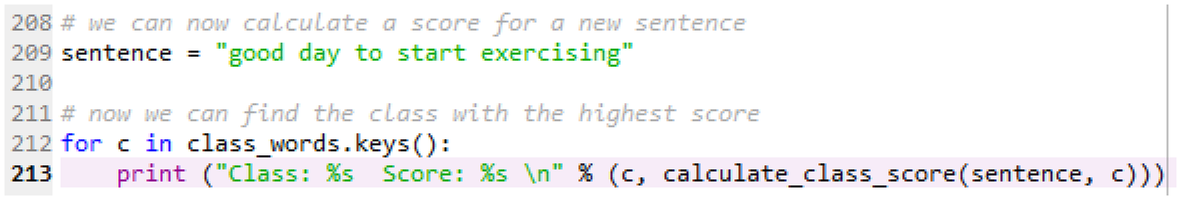
Step 4 (developing the classifier algorithm): Our next step is to code our algorithm; our first version treated each word with equal weight.



```python
# calculate a score for a given class
def calculate_class_score(sentence, class_name, show_details=True):
    score = 0
    # tokenize each word in our new sentence
    for word in nltk.word_tokenize(sentence):
        # check to see if the stem of the word is in any of our classes
        if stemmer.stem(word.lower()) in class_words[class_name]:
            # treat each word with same weight
            score += 1

            if show_details:
                print ("   match: %s" % stemmer.stem(word.lower() ))
    return score
```

Figure 14: Handling scores

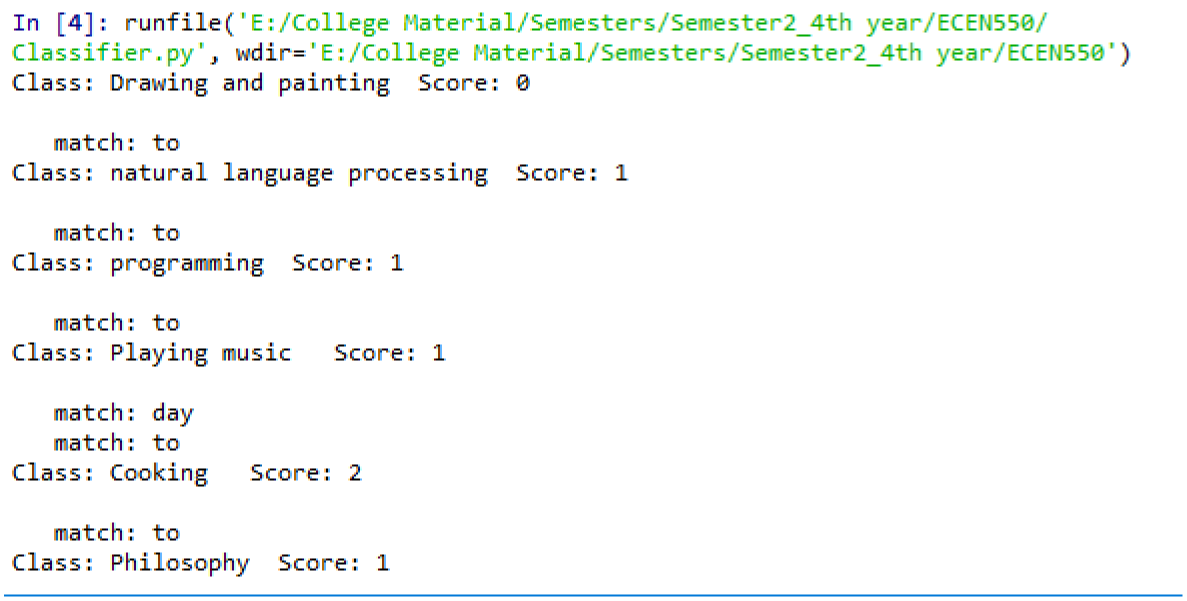
Each word from the given sentence is tokenized, stemmed, and lowercased, this is consistent with the transforms we applied to the corpus data. In other words: our input data is transformed consistently with our training data.



```python
# we can now calculate a score for a new sentence
sentence = "good day to start exercising"

# now we can find the class with the highest score
for c in class_words.keys():
    print ("Class: %s  Score: %s \n" % (c, calculate_class_score(sentence, c)))
```

Figure 15: Handling classes

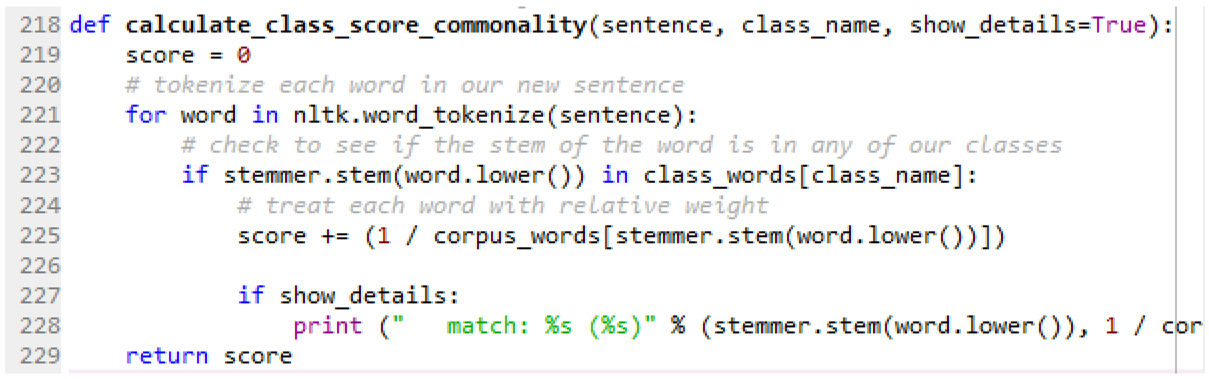
Classifying the sentence “good day to start exercising?” with the previous algorithm results in:



```
In [4]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/
Classifier.py', wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
Class: Drawing and painting  Score: 0

   match: to
Class: natural language processing  Score: 1

   match: to
Class: programming  Score: 1

   match: to
Class: Playing music   Score: 1

   match: day
   match: to
Class: Cooking   Score: 2

   match: to
Class: Philosophy  Score: 1
```

Figure 16: Testing scores

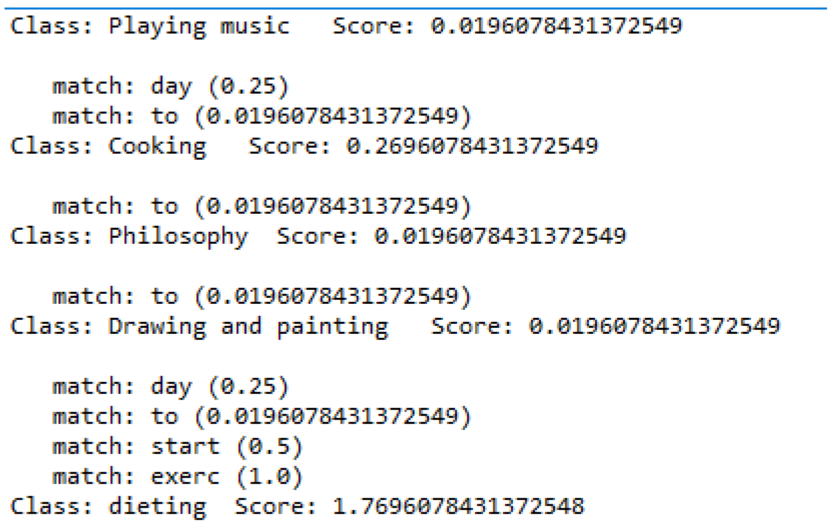
Notice each class generates a total score for the # of words that match. We can significantly improve our algorithm by accounting for the commonality of each word. The word "day" should carry a lower weight than the word "exercising" in most cases, because it is more common.



```
def calculate_class_score_commonality(sentence, class_name, show_details=True):
    score = 0
    # tokenize each word in our new sentence
    for word in nltk.word_tokenize(sentence):
        # check to see if the stem of the word is in any of our classes
        if stemmer.stem(word.lower()) in class_words[class_name]:
            # treat each word with relative weight
            score += (1 / corpus_words[stemmer.stem(word.lower())])

            if show_details:
                print ("   match: %s (%s)" % (stemmer.stem(word.lower()), 1 / cor
    return score
```

Figure 17: Handling Scores

To see the difference with the new suggested algorithm, we classify the sentence "good day to start exercising?" again. The results are much better, because "day" and "start" are more common terms (in this corpus) they carry lower weight and help to distinguish the correct class.

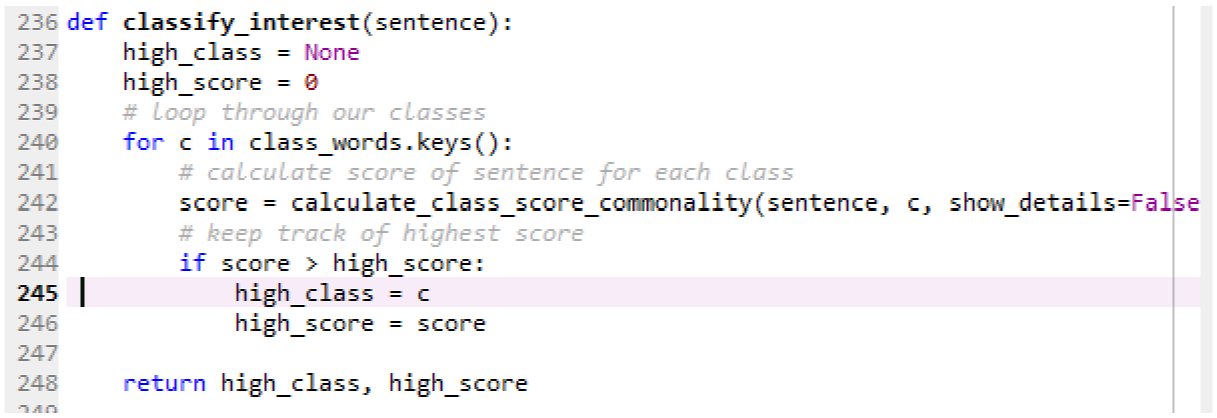


```
Class: Playing music   Score: 0.0196078431372549

   match: day (0.25)
   match: to (0.0196078431372549)
Class: Cooking   Score: 0.2696078431372549

   match: to (0.0196078431372549)
Class: Philosophy  Score: 0.0196078431372549

   match: to (0.0196078431372549)
Class: Drawing and painting   Score: 0.0196078431372549

   match: day (0.25)
   match: to (0.0196078431372549)
   match: start (0.5)
   match: exerc (1.0)
Class: dieting  Score: 1.7696078431372548
```

Figure 18: Score outputs

We can see here the correct class is dieting as it has the highest score.

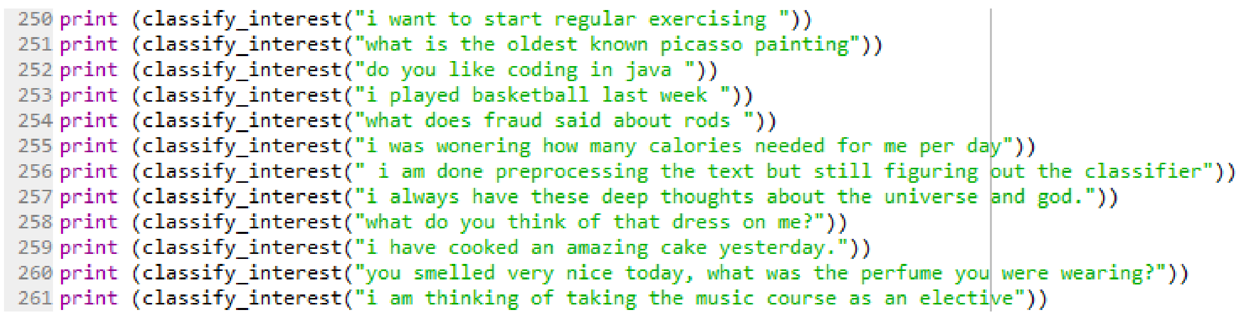
Step 5 (the final algorithm): Our last step is to abstract the algorithm, so it can be used simply.



```python
def classify_interest(sentence):
    high_class = None
    high_score = 0
    # loop through our classes
    for c in class_words.keys():
        # calculate score of sentence for each class
        score = calculate_class_score_commonality(sentence, c, show_details=False
        # keep track of highest score
        if score > high_score:
            high_class = c
            high_score = score

    return high_class, high_score
```

Figure 19: Classification

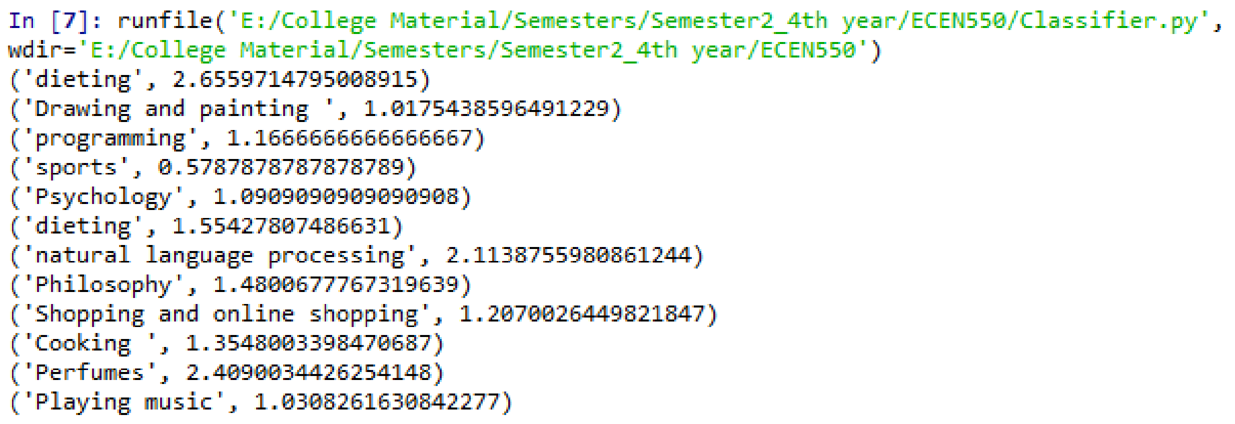
And now we can test our classifier with a few example inputs.



```python
print (classify_interest("i want to start regular exercising "))
print (classify_interest("what is the oldest known picasso painting"))
print (classify_interest("do you like coding in java "))
print (classify_interest("i played basketball last week "))
print (classify_interest("what does fraud said about rods "))
print (classify_interest("i was wonering how many calories needed for me per day"))
print (classify_interest(" i am done preprocessing the text but still figuring out the classifier"))
print (classify_interest("i always have these deep thoughts about the universe and god."))
print (classify_interest("what do you think of that dress on me?"))
print (classify_interest("i have cooked an amazing cake yesterday."))
print (classify_interest("you smelled very nice today, what was the perfume you were wearing?"))
print (classify_interest("i am thinking of taking the music course as an elective"))
```

Figure 20: Testing Classification

And the result of classification.



```
In [7]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/Classifier.py',
wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
('dieting', 2.6559714795008915)
('Drawing and painting ', 1.0175438596491229)
('programming', 1.1666666666666667)
('sports', 0.5787878787878789)
('Psychology', 1.0909090909090908)
('dieting', 1.55427807486631)
('natural language processing', 2.1138755980861244)
('Philosophy', 1.4800677767319639)
('Shopping and online shopping', 1.2070026449821847)
('Cooking ', 1.3548003398470687)
('Perfumes', 2.4090034426254148)
('Playing music', 1.0308261630842277)
```

Figure 21: Final Output

6. Testing

The dataset used in training the Naïve Bays classifier is presented. The dataset contains 132 sentences with 12 classes or interests. This dataset is developed for the purpose of our project, it can grow more in the future to include more interests and more sentences relevant to each interest to enable the classifier to be more accurate in extracting user's interests. The 12 interests are: sports, dieting, programming, Natural Language Processing, online courses, psychology, philosophy, drawing and painting, playing music, shopping and online shopping, perfumes, and cooking.

1. Sports
    - Team sports are probably one of the most popular kinds of sports all over the world, and the most famous and highest-paid athletes usually come from sports teams.
    - Team sports include basketball, football, soccer, hockey, and volleyball. Among these, basketball, football and soccer account for the largest teams in the world.
    - Famous leagues include the NBA for basketball, the NFL for football, and the Premiere League and UEFA Champion's League for soccer.
    - These are sports such as tennis, archery, gymnastics and running that are usually played by a single individual.
    - Although there are teams depending on the tournament, which usually happens during the Olympics, for the most part, these kinds of sports are played by an individual against another person, sometimes even belonging from the same team as they are.
2. Dieting
    - By eating a balanced diet, people can get the nutrients and calories they need and avoid eating junk food, or food without nutritional value.

- A healthful, balanced diet includes foods from these five groups: Vegetables, fruits, grains, protein, dairy.
- A poor diet is a common reason why people struggle with weight loss.
- When combined with a regular exercise routine, a balanced diet can help a person reduce their risk factors for obesity or gaining weight.

3. Programming
   - Most popular programming languages are C, C++, C#, JAVA, Python, JavaScript /TypeScript, Objective-C, PHP, Ruby, Scala.
   - How are you going to "choose a language" for development on iPhone, if there are only two of them?
   - How are you going to "choose a language" for development on Android, if there is only "one of them"?
   - There is still some kind of a choice in Web-development.
4. Natural language processing
   - Natural Language Processing is the technology used to aid computers to understand the human's natural language.
   - It's not an easy task teaching machines to understand how we communicate.
   - A human talk to the machine.
   - The machine captures the audio.
   - Audio to text conversion takes place.
   - Processing of the text's data.
   - Data to audio conversion takes place.
   - The machine responds to the human by playing the audio file.
   - Language translation applications such as Google Translate.
   - Word Processors such as Microsoft Word and Grammarly that employ NLP to check grammatical accuracy of texts.
   - Interactive Voice Response (IVR) applications used in call centers to respond to certain users' requests.
   - Personal assistant applications such as OK Google, Siri, Cortana, and Alexa.
   - Here are some syntax techniques that can be used: Morphological segmentation, Word segmentation, Part-of-speech tagging, Parsing, tokenizing, Stemming, Lemmatization.
5. Online courses

- Famous platforms for online courses are: Udemy, Teachable, THINKIFIC, Learnworldss, RUZUKU, PODIA, TEACHABLE, Academy of mine, cousera, EDX, treehouse.

6. Psychology

- The Beginnings of Psychology as a Discipline.
- Psychology perspectives are behaviorism, Psychodynamic approach, humanistic approach, cognitive psychology, biological approach.
- Psychology is the study of the mind, how it works, and how it affects behavior.
- There are different types of psychology, such as clinical, health, occupational, cognitive, forensic, social, and developmental psychology.
- A person with a condition that affects their mental health may benefit from assessment and treatment with a psychologist.
- A psychologist may offer treatment that focuses on behavioral adaptations.
- A psychiatrist is a medical doctor who is more likely to focus on medical management of mental health issues.
- It’s the most interesting because it’s about us, it’s about the most important and intimate aspects of our lives, it’s about language and perception, it’s about our memory of things, it’s about our dreams, love, hate, it’s about morality, our sense of right and wrong, it’s about when things go wrong as in depression, our anxiety, it’s about happiness.

7. Philosophy

- History is Philosophy teaching by examples.
- Metaphysics is a dark ocean without shores or lighthouse, strewn with many a philosophic wreck.
- Science is what you know. Philosophy is what you don't know.
- here is only one thing a philosopher can be relied upon to do, and that is to contradict other philosophers.
- If God did not exist, it would be necessary to invent Him.
- Socrates, William of Ockham, Thomas Hobbes, René Descartes, Bishop George Berkeley.
- Does God have supreme power?
- Will the world be a better place if caste and religion cease to exist?
- What is the meaning of true love?
- Why do we strive for perfection if it is not attainable?

- Does free will exist, or is every action predetermined?
- What is human consciousness?
- Why do we do things we do not like to do?
- Do atheists make their own gods?
- Can artificial intelligence be creative?
- If judgment is for God, why do we pass judgment?
- Can religious beliefs affect scientific thinking?

8. Drawing and painting
    - HOW TO DRAW & PAINT FASTER?
    - Are you struggling to get your Art projects done on time? Some students – even those who are dedicated and hard-working – find it challenging to work at the pace required in a Visual Art course.
9. Playing music
    - I have always maintained that learning how to play an instrument is beneficial.
    - Have you ever felt chills down your spine while listening to music?
    - How powerful the effects of music, though, depends on your personality.
    - Music has impacted me… helping my ability to do math and to read, and to think critically.
    - When you play a musical instrument, you have to learn about tone and about scores and your ability to store audio information becomes better.
    - Popular Woodwind Instruments: Flute, Piccolo, Shakuhachi, Clarinet, Bassoon, Conch, English horn, Oboe, Saxophone, Shehnai, Bagpipe, Pianica, Harmonica.
    - Popular Percussion Instruments: Drum, Congo, Djembe, Duff, Tabla, Dhol, Nagara, Cymbals, Bells, Xylophone, Marimba, Triangle, Tambourine.
    - Popular Brass Instruments: Trumpet, Trombone, Bugle, French horn, Tuba, Alto Horn, Bazooka, Cimbasso, Flatt trumpet.
    - Popular String Instruments: Guitar, Violin, Viola, Sitar, Cello, Double Bass, Mandolin, Banjo, Harp, Sarod, Santoor.
    - Popular Electronic Instruments: Piano keyboards, Octopads, Rhythm machines, Samplers, Synthesizers, Synclavier, Theremin, Eigenharp, Mellotron, Omnichord.
10. Shopping and online shopping
    - Here are some shopping tips to help you get the most from the money you spend.
    - Keep your receipts.
    - Don’t use a credit card if you can’t afford the price.

- Try before you buy.
- Ask family and friends.
- Confirm the full price.
- Watch for sales, coupons and rebates.
- Consider negotiating.
- Find the best overall value — quality, service and price.
- Inspect products before you buy.
- Understand the warranty.
- Know the return policy.
- Save your receipt.
- Take advantage of membership discounts.
- When you shop online, you have to start by searching for a product. This can be done by visiting a store's website.
- Some sites like Amazon and Yahoo! make it possible for them to display products or build their own online stores for a monthly fee.
- Other websites like eBay and Bidz provide an auction format, in which sellers can display items for a minimum price and buyers can bid on these items until the listing ends or the seller chooses to award it to a buyer.

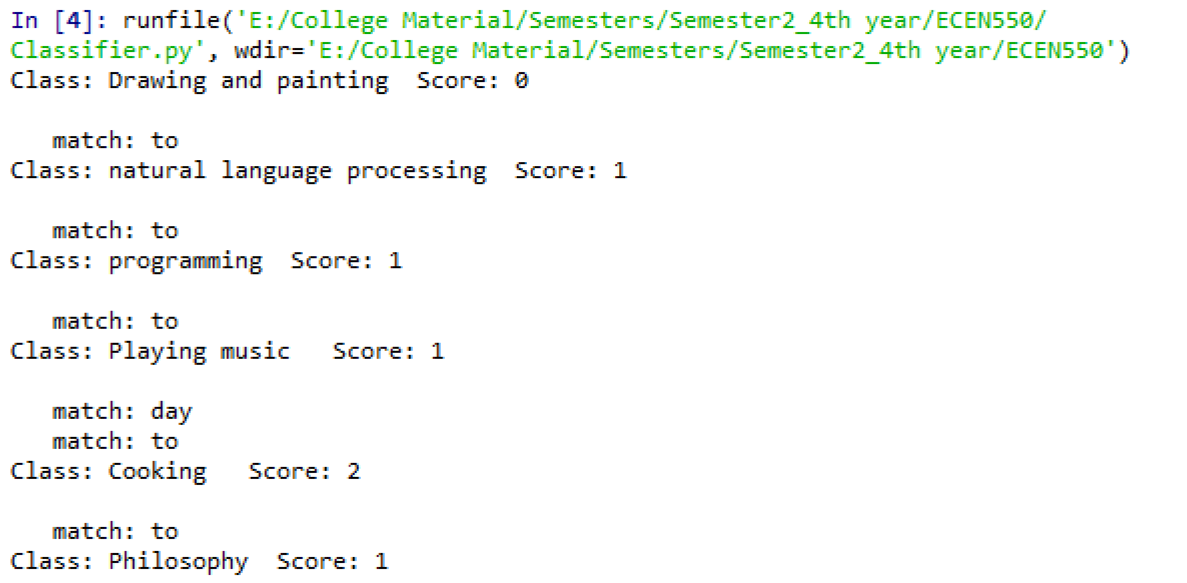
11. Perfumes
    - How to Choose the Right Perfume or Cologne.
    - Comforting fragrance for cold winter days.
    - Is body odor a big turn off for you.
    - Do you like to wear your favorite perfume for that crucial meeting?
    - Well, if you can relate to the above situations, it means you love fragrance.
    - Perfumes and deodorants are popular today.
12. Cooking
    - I love cooking.
    - I am following some amazing cooking recipes.
    - The kitchen not only taught me the art of food, but also prepared me to be a better developer.
    - Orepare home-cooked meals can seem like a daunting task.
    - Food brings people together and cooking at home is a great way to unite your family over the dining table.
    - It is a passion of mine. Having spent almost 25 years in kitchens and catering events.

- Personally, I love to cook and bake.
- There is something magical about taking some flour, yeast, salt and water and coming out with a delicious, sustaining bread.
- Taking day old mashed potatoes and leftover meatloaf to make Potato Balls.
- Hunting through the refrigerator, collecting the ingredients, preparing them and putting them together to make something delicious.
- Seeing how different veggies, meat and spices combine to give those gorgeous flavors, makes you really happy.

The testing procedure used are divided into 2:

1. Treating each word with equal weight.
2. Accounting for the commonality of each word.

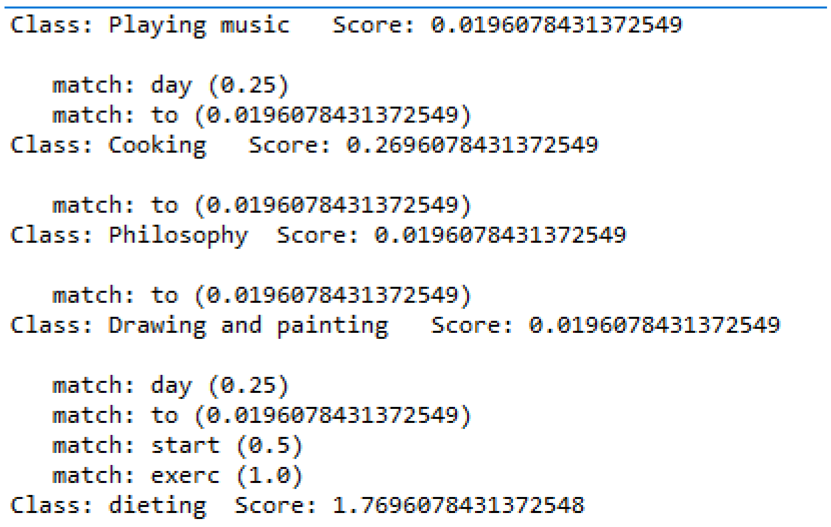
The first procedure produced poor results as shown in figure (). By this procedure, multiple classes will be assigned to the test sentence. The test sentence is “good day to start exercising?”. The highest score was 2 for class “cooking” which is wrong while the correct class is “dieting”.



```
In [4]: runfile('E:/College Material/Semesters/Semester2_4th year/ECEN550/
Classifier.py', wdir='E:/College Material/Semesters/Semester2_4th year/ECEN550')
Class: Drawing and painting  Score: 0

   match: to
Class: natural language processing  Score: 1

   match: to
Class: programming  Score: 1

   match: to
Class: Playing music   Score: 1

   match: day
   match: to
Class: Cooking   Score: 2

   match: to
Class: Philosophy  Score: 1
```

Figure 22: Why scores were fixed

The second procedure produced very accurate results, this procedure is directly proportional to the size and data contained in the dataset. As the dataset increases and covers more relevant sentences, the output of this procedure becomes very accurate. See results for the same test sentence in figure (). Notice that the highest score was 1.76 for class “dieting” which is the desired output.



```
Class: Playing music   Score: 0.0196078431372549

   match: day (0.25)
   match: to (0.0196078431372549)
Class: Cooking   Score: 0.2696078431372549

   match: to (0.0196078431372549)
Class: Philosophy  Score: 0.0196078431372549

   match: to (0.0196078431372549)
Class: Drawing and painting   Score: 0.0196078431372549

   match: day (0.25)
   match: to (0.0196078431372549)
   match: start (0.5)
   match: exerc (1.0)
Class: dieting  Score: 1.7696078431372548
```

Figure 23: Desired output

7. <u>Conclusion</u>

in a nutshell, the program works just a planned and only requires refactoring or enhancement, which will be discussed in the future work segment. all it does is extract text from speech via Google's API for speech recognition, then it preprocess the text based on natural language processing techniques to prepare the text for classification via Naive Bayes, which finally returns the user's interest based on speech and its calculated score towards the class in interest.

8. <u>Future Work</u>

The goal set regarding interest extraction has been met; however not as efficiently as planned. some of the main fixes for this project include, but not are not limited to, refactoring code to train classifiers using larger datasets, using machine learning, saving classifiers, and increasing runtime efficiency.

References

[1] Gk_. (2017, January 06). Soul of the Machine: How Chatbots Work. Retrieved from https://medium.com/@gk_/how-chat-bots-work-dfff656a35e2

[2] 10154238595463041. (2017, January 11). Text Classification using Algorithms. Retrieved from https://chatbotslife.com/text-classification-using-algorithms-e4d50dcba45

[3] Natural Language Processing with IBM Watson and Stanford NLP. (2018, February 15). Retrieved from https://zerone-consulting.blog/2017/12/06/natural-language-processing-with-ibm-watson-and-stanford-nlp/

[4] Pre-Processing in Natural Language Machine Learning. Fortney, K. (2017). Retrieved from https://towardsdatascience.com/pre-processing-in-natural-language-machine-learning-898a84b8bd47