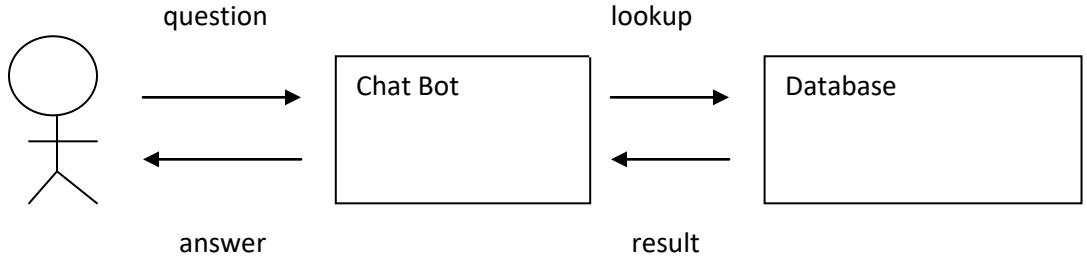
Natural Language Processing involves translation written sentences into meaning. A good example is a Chatbot. A Chatbot is also known as a digital assistant. Chat box are used every where to answer questions from customers and provide accurate answers. When you phone your bank you can ask for your bank account balance and a digital assistant will answer you with the correct balance. You could also ask out the digital assistant to go out for lunch, but they might reply they are not available today but may be tomorrow. A Chatbot interacts with a person , picks out important words, queries a data base to give the correct response.



The database connected to a chatbot could return Reponses for typical responses or more accurate responses for actual questions, like the balance on somebody's bank account.

Chatbots use NLP (Natural Language Processing) to analyze, understand, and derive meaning from human language.

PreProcessing Text

In order to process text for a computer we must pre-process the text as follows:

pre-processing includes:

- convert the text into **lowercase**
- **Word Tokenizing** converting the text strings into a list of tokens
- **Sentence Tokenizing** converting the sentence text into a list of sentences
- Remove **Punctuation**
- Remove **Stop words,** words that are common like at, this, a and an that are not needed
- **Stem** words to their base form known as **Stemming.**

  if the words end s in s remove the ‘s’: runs -> run
  if the word ends in 'ed', remove the 'ed': stopped -> stop
  if the word ends in 'ing', remove the 'ing': running -> run
  if the word ends in 'ly', remove the 'ly': slowly -> slow

- **Lemmatization**

  Lemmatization is similar to Stemming it links words with similar meaning to one word. Stemming converts words to their base where as **Lemmatization** converts words to a base meaning. Results For Lemmatization and Stemming may different or the same depending on the words encountered.

```
Examples of lemmatization:

        rocks : rock
        corpora : corpus
        better : good
        running : run
        ran : run
```

```
Examples of Stemming:

        rocks : rock
        corpora : corpus
        better : good
        running : run
        ran : ran
```

Term Frequency-Inverse Document Frequency TF-IDF

Term Frequency: is a scoring of the frequency of the word in the current document.
Tf-idf weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. A Corpus is a language resource consisting of a large and structured set of texts.

$$TF = \frac{\text{Number of times A term appears in a document}}{\text{Number of terms in the document}}$$

Inverse Document Frequency: is a scoring of how rare the word is across documents.

$$IDF = 1+\log(N/n)$$

Where:
N is the number of documents
n is the number of documents a term has appeared in.

Example calculation:

We have a document with 100 words and that the words apple

$$TF = \frac{\text{Number of times a term appears in a document}}{\text{Number of terms in the document}} = \frac{5}{100} = .05$$

If we have 5 documents

$$IDF = 1+\log(N/n) = 1 + \log(5/.05) = 1 + \log(100) = 1 + 2 = 3$$

Cosine Similarity

TF-IDF is a transformation applied to texts to get two real-valued vectors in vector space. **Cosine similarity** is a measure of similarity between two non-zero vectors. We can then obtain the **Cosine** similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors.

Using this formula, we can find out the similarity between any two documents d1 and d2.

$$\text{Cosine Similarity (d1, d2)} = \frac{\text{Dot product(d1, d2)}}{||d1|| * ||d2||}$$

where d1,d2 are two non zero vectors.

NATURAL language processing using NTLK

The NLTK (Natural Language Toolkit) is a platform for building Python programs to work with human language.

To install in your python:

```
pip install nltk
```

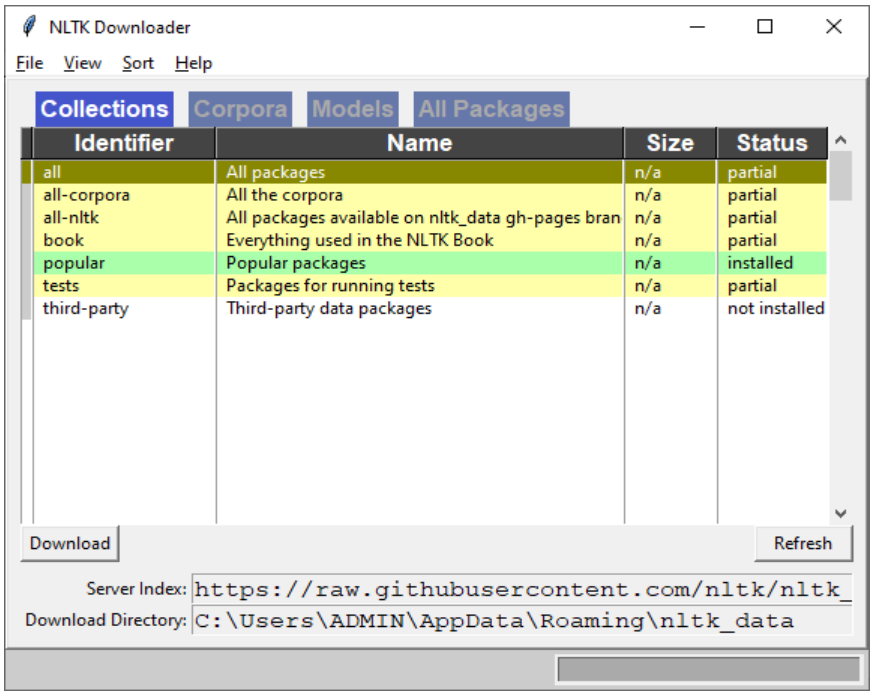
To run in your program:

```
import nltk
```

All the NTLK modules are available to down using the following NTLK module down loader.

```
nltk.download()
```

You can choose the modules you want to download, but for these lessons you do not need to use the downloader. It will take a long time to down load everything from nltk, so you should not do it.



Sentence tokenize

The NLTK library has the **sent_tokenize** function to convert text into a list of sentences.

Example Sentence tokenizing using these test sentences:

text = "I was slowly taking a ride in the car that suddenly stopped. I was slowly riding in a car that suddenly stopped."

You will first need to download the punkt resource.

```
nltk.download('punkt')
```

Once loaded into you system you do not need to load again

You first need to convert all text to lower case

```
text = text.lower()
print(text)
```

Next we tokenize the text into a list of sentences

```
from nltk.tokenize import sent_tokenize
sentences  = nltk.sent_tokenize(text)

print("sentence tokens:")
print(sentence_tokens)
```

You should get something like this:

```
['i was slowly taking a ride in the car that suddenly stopped.',
 'i was slowly riding in a car that suddenly stopped.']
```

Remove punctuation

To remove punctuation we make dictionary of punctuation letters where the key are the punctuation letters and the value are empty strings.

```
import string
punct_dict = dict((ord(punct), None) for punct in string.punctuation)
```

Next we use the python **translate** string method to remove all punctuation

```
text=text.translate(punct_dict)
print(text)
```

You should get something like this

```
i was slowly taking a ride in the car that suddenly stopped i was slowly riding in a car that
suddenly stopped
```

Word tokenize

The NLTK library has the word_tokenize() function to convert a sentence to tokens.

You will first need to download the punkt resource if you have not already did so. Once loaded into you system you do not need to load again

```
from nltk.tokenize import word_tokenize
tokens = nltk.word_tokenize(text)
print("word tokens:")
print(word_tokens)
```

You should get something like this:

```
['i', 'was', 'slowly', 'taking', 'a', 'ride', 'in', 'the', 'car', 'that', 'suddenly', 'stopped', 'i', 'was',
'slowly', 'riding', 'in', 'a', 'car', 'that', 'suddenly', 'stopped']
```

Note: Before word tokenizing we have already removed punctuation. The word_tokenize does not simply split a string based on whitespace, but also separates punctuation into tokens.

remove stopwords

Stop words are common words like at, this, a and an that are not needed. You first need to down load the stop words resource.

```
nltk.download('stopwords')
```

You only need to do this once. Once that are in your system you do not need to do this again.

Once you have loaded in the stopwords resource you can get a list of stop words for your language and then use these stop words to remove the stop words from your word list.

```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
word_tokens_stop = [w for w in word_tokens if not w in stop_words]
print("word tokens stop:")
print(word_tokens_stop)
```

You should get something like this:

```
['slowly', 'taking', 'ride', 'car', 'suddenly', 'stopped', 'slowly', 'riding', 'car',
'suddenly', 'stopped']
```

stemming

Stemming converts convert words into their base form

You first need to make a stemmer , we are using the **PorterStemmer** from the nltk library.

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

Once you got the stemmer you use the stemmer on each word in the word list

```
stemmed_words = [stemmer.stem(w) for w in word_tokens_stop]
print("stemmed words")
print(stemmed_words)
```

You should get something like this:

```
['slowli', 'take', 'ride', 'car', 'suddenli', 'stop', 'slowli', 'ride', 'car',
'suddenli', 'stop']
```

We can make a table to Compare actual word to stemmed word

Actual	**Stemmed**
Slowly	Slowly
Taking	Take
Ride	Ride
Car	Car
Suddenly	Suddenly
Stopped	Stop
Slowly	Slowly
Riding	Ride
Car	Car
Suddenly	Suddenly
Stopped	Stop

lemmatization

Lemmatization normalizes a word based on the context and vocabulary of the text. In NLTK, you can lemmatize sentences using the WordNetLemmatizer class.

First, you need to download the wordnet resource from the NLTK downloader in the Python terminal.

```
nltk.download('wordnet')
```

wordnet is a semantically-oriented dictionary of English
once you down loaded wordnet into your system you do not need to load it in again

Next we import the WordNetLemmatizer class and make a WordNetLemmatizer object.

```
from nltk.stem.wordnet import WordNetLemmatizer
len = WordNetLemmatizer()
```

Sometimes, the same word can have a multiple lemmas based on the meaning / context. We need to determine the parts of speech for each word POS if is a noun or verb.

Lemmatizing can lemmatize a word to a verb using the ‘v’ parameter or a noun using the ‘n’ parameter or other parts of speech.

```
lemmed_words_verb = [lemmer.lemmatize(w,'v') for w in word_tokens_stop]
print("lemmed words verb")
print(lemmed_words_verb)

lemmed_words_noun = [lemmer.lemmatize(w,'n') for w in word_tokens_stop]
print("stemmed words noun")
print(lemmed_words_noun)
```

You should get something like this:

```
lemmed words verb
['slowly', 'take', 'ride', 'car', 'suddenly', 'stop', 'slowly', 'rid', 'car', 'suddenly', 'stop']

stemmed words noun
['slowly', 'taking', 'ride', 'car', 'suddenly', 'stopped', 'slowly', 'riding', 'car', 'suddenly',
'stopped']
```

It may not be possible manually provide the current Parts of speech tag POS tag for every word for large texts. So, instead, we will find out the correct POS tag for each word, map it to the right input character that the WordnetLemmatizer accepts and pass it as the second argument to lemmatize().

We can make a table to Compare actual word to stemmed word

Actual	Lemmed verb	Lemmed noun
Slowly	slowly	slowly
Taking	take	taking
Ride	ride	Ride
Car	car	Car
Suddenly	suddenly	suddenly
Stopped	stop	stopped
Slowly	slowly	slowly
Riding	rid	Ride
Car	car	Car
Suddenly	suddenly	suddenly
Stopped	stop	stopped

Determining parts of speech POS

In nltk, it is available through the nltk.pos_tag() method. It accepts only a list (list of words), even if its a single word.

```
# pos
pos = nltk.pos_tag(word_tokens_stop)
print("POS")
print(pos)
```

```
POS
[('slowly', 'RB'), ('taking', 'VBG'), ('ride', 'NN'), ('car', 'NN'), ('suddenly', 'RB'), ('stopped', 'VBD'),
('slowly', 'RB'), ('riding', 'VBG'), ('car', 'NN'), ('suddenly', 'RB'), ('stopped', 'VBD')]
```

Here is a table of the parts of speech

```
POS tag list:

CC      coordinating conjunction
CD      cardinal digit
DT      determiner
EX      existential there (like: "there is" ... think as "there exists")
FW      foreign word
IN      preposition/subordinating conjunction
JJ      adjective       'big'
JJR     adjective, comparative 'bigger'
JJS     adjective, superlative 'biggest'
LS      list marker     1)
MD      modal   could, will
NN      noun, singular 'desk'
NNS     noun plural     'desks'
NNP     proper noun, singular  'Harrison'
NNPS    proper noun, plural    'Americans'
PDT     predeterminer  'all the kids'
POS     possessive ending      parent\'s
PRP     personal pronoun       I, he, she
PRP$    possessive pronoun     my, his, hers
RB      adverb  very, silently,
RBR     adverb, comparative    better
RBS     adverb, superlative    best
RP      particle       give up
TO      to      go 'to' the store.
UH      interjection   errrrrrrrm
VB      verb, base formtake
VBD     verb, past tense       took
VBG     verb, gerund/present participle       taking
VBN     verb, past participle  taken
VBP     verb, sing. present, non-3d    take
VBZ     verb, 3rd person sing. present takes
WDT     wh-determiner  which
WP      wh-pronoun     who, what
WP$     possessive wh-pronoun  whose
WRB     wh-abverb      where, when
```

We can now decide which word is a noun, verb, adjective or adverb.

we use the following function to determine the parts of speech where the noun is the default.

```
def get_wordnet_pos(word):

   # Map POS tag to first character
   tag = nltk.pos_tag([word])[0][1][0].upper()

   # make dictionary of tags
   tag_dict = {"J": wordnet.ADJ,
           "N": wordnet.NOUN,
           "V": wordnet.VERB,
           "R": wordnet.ADV}

   # return pos_tag default is wordnet.NOUN
  return tag_dict.get(tag, wordnet.NOUN)
```

We call the function on each word

```
lemmed_words = [lemmer.lemmatize(w,get_wordnet_pos(w)) for w in word_tokens_stop]
```

You should get something like this:

```
['slowly', 'take', 'ride', 'car', 'suddenly', 'stop', 'slowly', 'rid', 'car', 'suddenly', 'stop']
```

Our table of lemmed words is now

We can make a table to Compare actual word to stemmed word

Actual	POS	
Slowly	RB	slowly
Taking	VBG	Take
Ride	NN	Ride
Car	NN	Car
suddenly	RB	suddenly
Stopped	VBD	Stop
Slowly	RB	slowly
Riding	VBG	Rid
Car	NN	Car
suddenly	RB	suddenly
Stopped	VBD	Stop

Frequency distribution

Nltk has the FreqDist function to print out the frequency distribution each word.

We first import the FreqDist module

```
from nltk import FreqDist
```

Then we calculate the frequency distribution of the lemmed words

```
# frequency distribution
Freq_dist = FreqDist(lemmed_words )
```

We just print out the first 10

```
print("frequency distribution")
print(freq_dist.most_common(10))
```

You should get something like this:

```
frequency distribution
[('slowly', 2), ('car', 2), ('suddenly', 2), ('stop', 2), ('take', 1), ('ride', 1), ('rid', 1)]
```

tdif

TF-IDF stands for “Term Frequency — Inverse Data Frequency”.

Term Frequency (tf): gives us the frequency of the word in each document in the corpus. It is the ratio of number of times the word appears in a document compared to the total number of words in that document. It increases as the number of occurrences of that word within the document increases. Each document has its own tf.

$$TF = \frac{\text{Number of times A term appears in a document}}{\text{Number of terms in the document}}$$

Inverse Data Frequency (idf): used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score. It is given by the equation below.

$$IDF = 1+\log(N/n)$$

Where:
N is the number of documents
n is the number of documents a term has appeared in.

Combining these two we come up with the TF-IDF score (w) for a word in a document in the corpus. It is the product of tf and idf:

TF-IDF = TF * IDF

We now calculate the TF-IDF for our two above lemmed sentences, assuming each in separate documents.

Sentence1: 'slowly', 'take', 'ride', 'car', 'suddenly', 'stop',

Sentence2: 'slowly', 'ride', 'car', 'suddenly', 'stop'

Calculating tf-idf of two sentences, (each in separate documents)

	TF			TF*IDF	
Word	A	B	IDF	A	B
Slowly	1/6	1/6	log(2/2)=0	0	0
Take	1/6	0/6	log(2/1)=.3	.05	0
Ride	1/6	1/6	log(2/2)=0	0	0
Car	1/6	1/6	log(2/2)=0	0	0
Suddenly	1/6	1/6	log(2/2)=0	0	0
Stop	1/6	1/6	log(2/2)=0	0	0
Rid	0/6	1/6	log(2/1)=.3	0	.05

From the above table, we can see that TF-IDF of common words was zero, which shows they are not significant. On the other hand, the TF-IDF of “take” and “rid” are non-zero. These words have more significance.

Using sklearn to calculate TF-IDF

Text Analysis is a major application field for machine learning algorithms. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

In order to address this, scikit-learn provides utilities for the most common ways to extract numerical features from text content, namely:

- **tokenizing** strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
- **counting** the occurrences of tokens in each document.
- **normalizing** and weighting with diminishing importance tokens that occur in the majority of samples / documents.

In this scheme, features and samples are defined as follows:

- each **individual token occurrence frequency** (normalized or not) is treated as a **feature**.
- the vector of all the token frequencies for a given **document** is considered a multivariate **sample**.

A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the **Bag of Words** or “Bag of n-grams” representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

Tf–idf term weighting

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

Sklern has the **TfidfVectorizer** module that convert a collection of raw documents to a matrix of TF-IDF features.

First, we will import TfidfVectorizer from sklearn.feature_extraction.text:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Now we make a TfidfVectorizer object using English stop words and then call fit_transform method to calculate the TF-IDF score for the text.

```
vectorizer = TfidfVectorizer(stop_words='english')
tfidf = vectorizer.fit_transform(sentence_tokens)
print("tfidf")
print(tfidf)
```

We also print out the word features

```
print("feature names")
print(vectorizer.get_feature_names())
```

You should get something like this:

```
tfidf
  (0, 4)   0.35464863330313684
  (0, 5)   0.35464863330313684
  (0, 0)   0.35464863330313684
  (0, 1)   0.49844627974580596
  (0, 6)   0.49844627974580596
  (0, 3)   0.35464863330313684
  (1, 2)   0.5749618667993135
  (1, 4)   0.40909010368335985
  (1, 5)   0.40909010368335985
  (1, 0)   0.40909010368335985
  (1, 3)   0.40909010368335985
feature names
['car', 'ride', 'riding', 'slowly', 'stopped', 'suddenly', 'taking']
```

Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors. We can then obtain the **Cosine** similarity of any pair of vectors by taking their dot product and dividing that by the product of their norms. That yields the cosine of the angle between the vectors.

Using this formula, we can find out the similarity between any two documents d1 and d2.

$$\text{Cosine Similarity (d1, d2)} = \frac{\text{Dot product(d1, d2)}}{||d1|| * ||d2||}$$

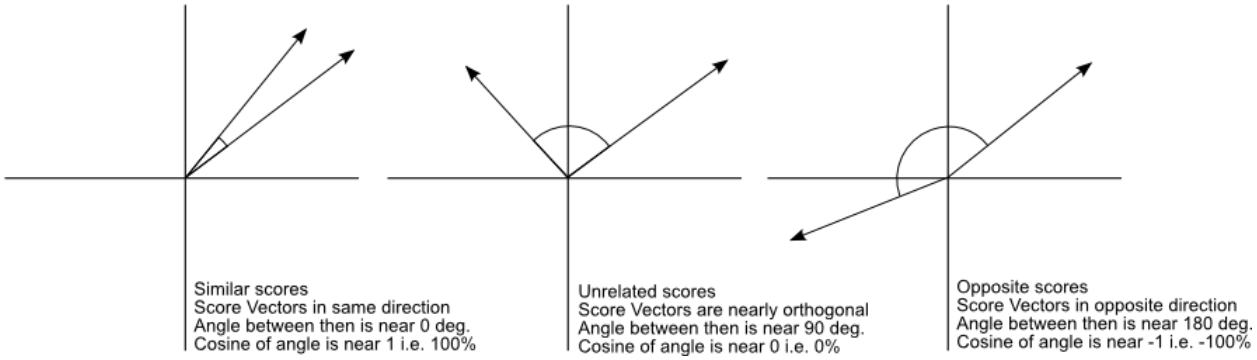
where d1,d2 are two non zero vectors.

Mathematically:

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

Cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space. The smaller the angle, higher the cosine similarity.



Calculating cosine similarity with sklearn

Sklearn has the **cosine_similarity** function to calculate Cosine similarity between tf-idf's.

We first import the **cosine_similarity** module from sklearn.metrics.pairwise

```
from sklearn.metrics.pairwise import cosine_similarity
```

We calculate cosine_similariry between the last tf-idf tfidf[-1] and the tf-idf from previous section.

```
print("tfidf[-1]")
print(tfidf[-1])
```

```
tfidf[-1]
  (0, 2)   0.5749618667993135
  (0, 4)   0.40909010368335985
  (0, 5)   0.40909010368335985
  (0, 0)   0.40909010368335985
  (0, 3)   0.40909010368335985
```

```
print("tfidf")
print(tfidf)
```

```
tfidf
  (0, 4)   0.35464863330313684
  (0, 5)   0.35464863330313684
  (0, 0)   0.35464863330313684
  (0, 1)   0.49844627974580596
  (0, 6)   0.49844627974580596
  (0, 3)   0.35464863330313684
  (1, 2)   0.5749618667993135
  (1, 4)   0.40909010368335985
  (1, 5)   0.40909010368335985
  (1, 0)   0.40909010368335985
  (1, 3)   0.40909010368335985
```

```
print("cosine_similarity")
print(cosine)
```

```
cosine_similarity
[[0.58033298 1.       ]]
```

The 0.58033298 represents the cosine similarity between the last td-idf compared to all the td-idf. The 1.0 is the td-idf itself.

We then extract the 1D array of values

```
values = cosine[-2]
print("values")
print(values)
```

```
values
[0.58033298 1.       ]
```

We then get a list of sorted indexes

```
indexes = values.argsort()
print("indexes")
print(indexes)
```

```
indexes
[0 1]
```

The first index represents the first document to the last document

Which is the first sentence in our list of sentences

```
Closest_index = indexes[-2]
```

```
0
```

We then flatten and sort the values

```
flat = values.flatten()
flat.sort()
```

```
[0.58033298 1.       ]
```

The closest tdif is the first one

```
req_tfidf = flat[-2]
print("req_tfidf")
 print(req_tfidf)
```

```
0.5803329846765686
```

We now have enough information to code our chatbot.

Natural Language Homework 1

Make a function called **get_lemmed_tokens(text):** that will convert al text to lower case, remove all punctuation, remove stop words and lemmatize using pos tags.

Test it like this

```
TfidfVector = TfidfVectorizer(stop_words='english')
```

Then calculate the cosine similarity and choose the best response.
Put your python code in a file called natural_language1.py

Chatbot Algorithm

The Chatbot algorithm is quite simple, we just get a text information document from the internet known as a corpus. We consider each sentence a separate document. We ask then user to type in a question. We attach the question to the end of our document. Then we lemmatize each word per sentence in the document. We then calculate the cosine similarity between the last tfidf and all the tfidf's. We then choose the best tfidf from the cosine similarity result. We print out the best sentence using the cosine similarity index. Example if we had a documents continuing information about fruits and we ask what is an apple, we could get the response an “apple is a delicious red fruit” , if we could not find a match and we could get the answers “I do not know the answer”.

Here are the steps:

Step 1:

Choose a corpus information document from the internet. The most popular one is the Chabot page on Wikipedia https://en.wikipedia.org/wiki/Chatbot.

Just put the text portion into a file called chstbot.txt.

Step 2:

Read in the file
convert to lower case

Step 3:

Tokenize sentences

Step 4:

Lemmatize words in sentences accordable to POS

Step 5:

Make some standard input and outputs sentences
standard_inputs = ("hello", "hi", "who are you","what is your name")
standard_responses = ("I am robo","How are you today")

Step 6:

Print a greeting

```
print("Hello, my name is Robo, I am a chat bot.")
print("Ask me questions about Chatbot's. To exit type Bye!: ")
```

Step 7:

Get user input
Of input one of the standard inputs respond with a random standard output
If input is bye then exit

Step 8:

Lemmatize user input and add the lemmatized sentences

Step 9:

Calculate tdidf from the lemmatized sentences

Step 10

Calculate cosine similarity

Step 11

Get a list of sorted cosine indexes
Store index of user cosine entry

Step 12

Get tfidf score
If the requested tfidf is 0 ask for more information
Else print out the sentence for the associated index

Here is the complete program

```
"""
chatbot.py

simple chatbot
"""

import nltk
import random

# needed to suppress warnings
import warnings
warnings.filterwarnings('ignore')


# download nltk modules
# nltk.download() # for nltk downloading packages (not needed)

nltk.download('punkt') # run once only
nltk.download('wordnet') # run once only
```

```
# get list of stop words
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')

# get punctuation string
import string
punct_dict = dict((ord(punct), None) for punct in string.punctuation)

# make lemmer
from nltk.corpus import wordnet
from nltk.stem.wordnet import WordNetLemmatizer
lemmer = WordNetLemmatizer()

# function to return POS for a word
def get_wordnet_pos(word):

    # Map POS tag to first character
    tag = nltk.pos_tag([word])[0][1][0].upper()

    # make dictionary of tags
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    # return pos_tag default is wordnet.NOUN
    return tag_dict.get(tag, wordnet.NOUN)

# function to return lemmed words of a text with POS
def get_lemmed_tokens(text):

    # to lower case
    text = text.lower()

    # remove punctuation
    text=text.translate(punct_dict)

    # make word tokens
    word_tokens = nltk.word_tokenize(text)

    # remove stop words
    word_tokens_stop = [w for w in word_tokens if not w in stop_words]
```

```
    # get lemmed tokens for POS
    lemmed_tokens = [lemmer.lemmatize(w,get_wordnet_pos(w)) for w in word_tokens_stop]

    # return lemmed words
    return " ".join(lemmed_tokens)


# make function to lemminize a sentence without POS
def lemminize(text):

    # change to lower case
    text = text.lower()

    # make a dictionary of punctuation
    punct_dict = dict((ord(punct), None) for punct in string.punctuation)

    # remove punctuation
    tokens = nltk.word_tokenize(text.translate(punct_dict))

    # leminize tokens
    lemmers = [lemmer.lemmatize(token) for token in tokens]

    return lemmers


# read in corpus
f=open('chatbot.txt','r',errors = 'ignore')
text=f.read()

# convert to lowercase
text = text.lower()

# get a list of sentence tokens
sent_tokens = nltk.sent_tokenize(text)

# lem sent tokens
lemmed_sent_tokens = []
for sent in sent_tokens:
    lemmed_tokens = get_lemmed_tokens(sent)
    lemmed_sent_tokens.append(lemmed_tokens)
```

```
# standaed inputs
standard_inputs = ("hello", "hi", "who are you","what is your name")
standard_responses = ("I am robo","How are you today")

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# return reponse from chatbot
def get_response():

   # make TfidfVectorizer
   #TfidfVec = TfidfVectorizer(tokenizer=lemminize, stop_words='english')
   TfidfVec = TfidfVectorizer(stop_words='english')

   # get tfidf's
   tfidf = TfidfVec.fit_transform(lemmed_sent_tokens)

   # calculate cosinse simularity of last tdif to all tdif
   cosines = cosine_similarity(tfidf[-1], tfidf)

   # get list of sorted indexes
   indexes=cosines.argsort()[0]

   # get index of reponse with greatest similarity
   index = indexes[-2]

   # get list of sorted cosines
   flat = cosines.flatten()
   flat.sort()

   # get reqest tdif
   req_tfidf = flat[-2]

   # check for a response
   if(req_tfidf==0):
      response = "I do not, know the answer, please give me more details:"
   else:
      response = sent_tokens[index]
   return response
```

```
# print greeing
print("Hello, my name is Robo, I am a chat bot.")
print("Ask me questions about Chatbot's. To exit type Bye!: ")

user = ""
while(user != 'bye'):
   user = input("> ").lower()

   # exit
   if user == 'bye' or user == 'goodbye' or user == 'exit'  or user == 'quit':
      print ("Have a nice day")

   # help
   elif user == 'help' or user == '?':
      print("Hello, my name is Robo, I am a chat bot.")
      print("Ask me questions about Chatbot's. To exit type Bye!: ")

   # general questions answered
   elif user in standard_inputs:
      print(random.choice(standard_responses))

   # respond to user
   else:

      # lem user
      lemmed_user = get_lemmed_tokens(user)

      # add user response to sentence tokens
      lemmed_sent_tokens.append(lemmed_user)

      # get response from chat bot
      response = get_response()
      print(response)

      # remove user response from sentencetokens
      lemmed_sent_tokens.remove(lemmed_user)
```

Running the program

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ADMIN\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Hello, my name is Robo, I am a chat bot.
Ask me questions about Chatbot's. To exit type Bye!:

> hello
I am robo

> what is a chatbox
I do not, know the answer, please give me more details:

> what is a chatbot?
chatbot
from wikipedia, the free encyclopedia

a virtual assistant chatbot

a chatbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, in lieu of providing direct contact with a live human agent.

> who is eliza?
development[edit]
among the most notable early chatbots are eliza (1966) and parry (1972).

> what is natural learning?
[37] these intelligent chatbots make use of all kinds of artificial intelligence like image moderation and natural language understanding (nlu), natural language generation (nlg), machine learning and deep learning.

> what is natural language processing?
[14]
one pertinent field of ai research is natural language processing.
```

> what is ai?
[14]
one pertinent field of ai research is natural language processing.

> bye
Have a nice day

Conclusion

The chatbot program is based on similarity matching. Although the answers are impressive it is not really answering our questions, just giving us a matching response, but it is a beginning in natural language processing. Out next Chatbot will be an AI Chatbot using neural networks to provide matching using categories.

AI Chatbot

The AI Chatbot uses a neural network to match known questions to known answers organized by categories.

Here are the steps to make the AI ChatBot

Step 1: pick categories

We first pick out some categories from our previous corpus document.

Overview
Background
Development
Applications
Limitations
Jobs

Step 2: make questions for answers

We then choose some important information and formulate some question to the provided information.

We start with each category, each category will get a few questions and a answers.

Overview

Question 1: What is a chat bot?

Answer 1: A **chatbot** is a software application used to conduct an on-line chat conversation

Question 2: What are the uses for a chat bot?

Answer 2: Chatbots are used in service, request routing, or information gathering

Question 3: Where are chatbots used?

Answer 3: Chat bots are used in e-commerce , education, entertainment, finance, health and news

Background

Question 1: What is Eliza?

Answer 1: involves the recognition of clue words or phrases, and the outputs corresponding pre-prepared or pre-programmed responses

Question 2: How does Eliza work?

Answer 2: it is actually based on rather simple pattern-matching

Development

Question 1: Who were the early charbots

Answer 1: Early chatbots are ELIZA (1966) and PARRY (1972) and later A.L.I.C.E., Jabberwacky and D.U.D.E

Question 2: How did the early chatbot's work

Answer 2: Purely based on pattern matching techniques without any reasoning capabilities

Applications

Question 1: What are applications for chatbots

Answer 1: Messaging apps, Company internal platforms, Customer Service, and Healthcare

Limitations

Question 1: What are the limitations of Chatbots?

Answer 1: efficiency highly depends on language processing , require a large amount of conversational data to train

Question 2: How does Eliza work?

Answer 2: it is actually based on rather simple pattern-matching

Jobs

Question 1: What jobs can a chatbot do?

Answer 1: often are used to automate tasks that do not require skill-based talents

Question 2: What jobs can a chatbot replace?

Answer 2: customer service, and call center workers and telemarketing.

Step 3: put categories, questions and answers in a file

We can put our categories, questions and answers in a file so we can update them.

Using the format:

Category
Question
Answer

Aichatbot.txt

Overview
What is a chat bot?
A chatbot is a software application used to conduct an on-line chat conversation
Overview
What are the uses for a chat bot?
Chatbots are used in service, request routing, or information gathering
Overview
Where are chatbots used?
Chat bots are used in e-commerce , education, entertainment, finance, health and news
Background
What is Eliza?
Involves the recognition of clue words or phrases t, and the outputs corresponding pre-prepared or pre-programmed responses
Background
How does Eliza work?
It is actually based on rather simple pattern-matching
Development
Who were the early charbots
Early chatbots are ELIZA (1966) and PARRY (1972) and later A.L.I.C.E., Jabberwacky and D.U.D.E
Development
How did the early chatbot's work
Purely based on pattern matching techniques without any reasoning capabilities
Applications
What are applications for chatbots
Messaging apps, Company internal platforms, Customer Service, and Healthcare
Limitations
What are the limitations of Chatbots?
Efficiency highly depends on language processing , require a large amount of conversational data to train
Limitations
How does Eliza work?
it is actually based on rather simple pattern-matching
Jobs
What jobs can a chatbot do?
Often are used to automate tasks that do not require skill-based talents
Jobs
What jobs can a chatbot replace?
Customer service, and call center workers and telemarketing.

Step 4: store information in lists

We read in the file of categories, questions and answers and put the information in lists.

```
# store info
categories = []
questions = []
answers = []

# open file
f=open("aichatbot.txt","r")

# read lines from file
line = f.readline().strip()
while(line):

   categories.append(line)
   line = f.readline().lower().strip()
   questions.append(line)
   line = f.readline().lower().strip()
   answers.append(line)
   line = f.readline().strip()
```

step 5 map question words to word list, map question to a category

Example mapping sentence to words:
if a sentence has a word in it from all the words then put a 1 else put a 0

	Word1 cat	Word2 dog	Word3 sat	Word 4 house	Word 5 rat	Word 6 bite	Word 7 ate
Sentence1 the cat sat on the rat	1	0	1	0	1	0	0
Sentence2 the cat bite the dog	1	1	0	0	0	1	0
Sentence3 the cat sat on the rat	1	0	1	0	1	0	0
Sentence4 the cat ate the rat	1	0	0	0	0	0	1

Example mapping sentence to a category:
For each sentence state the category

	Category 1 sitting	Category 2 bitting	Category 3 eating	Category 4 running
Sentence1 the cat sat on the rat	1	0	0	0
Sentence2 the cat bite the dog	0	1	0	0
Sentence3 the cat ran after the rat	1	0	0	1
Sentence4 the cat ate the rat	0	0	1	0

```
# stem word tokens
stemmed_words = [stemmer.stem(w) for w in word_tokens_stop]
stemmed_words = sorted(list(set(stemmed_words)))

# sort categories
sorted_categories = sorted(list(set(categories)))

training = []
output = []

# map questions to words
for i, question in enumerate(questions_tokenized_stopped):

    training_row = []

    # stem tokens
    stemmed_question = [stemmer.stem(token) for token in question]

    # check foe matched word
    for w in stemmed_words:
        if w in stemmed_question:
            training_row.append(1)
        else:
            training_row.append(0)
```

```
# map question to category
output_row = [0] * len(sorted_categories)
output_row[sorted_categories.index(categories[i])] = 1

training.append(training_row)
output.append(output_row)
```

Step 6: Setup neural network

We are using tensorflow that works quite fast. We have a input size for the number of words in the word list. We have 2 hidden layer each of 8 nodes. We have output node of size of the categories. The output layer use “softmax” where the output follows the input positively.

```
# neural net input and outputs
training = numpy.array(training)
output = numpy.array(output)

#tensorflow.reset_default_graph()
ops.reset_default_graph()

# make neural net
net = tflearn.input_data(shape=[None, len(training[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(output[0]), activation="softmax")
net = tflearn.regression(net)

# make neural network model
model = tflearn.DNN(net)

# train neural network
model.fit(training, output, n_epoch=1000, batch_size=8, show_metric=True)
```

Step 7 make prediction

Each query is mapped the question words and the sent to the neural network for category prediction. From the prediction results one is chosen randomly.

```
# get response from neural network
def get_response(query, words):

    # empty row
    row = [0] * len(words)

    # to lower case
    query = query.lower()

    # remove punctuation
    query=query.translate(punct_dict)

    # word tokenize question
    tokens = nltk.word_tokenize(query)

    #remove stop words
    tokens_stop = [w for w in tokens if not w in stop_words]

    # stem words
    stemmed_tokens = [stemmer.stem(word) for word in tokens_stop]

    # map query to question words
    for stemmed_word in stemmed_tokens:
        for i, w in enumerate(words):
            if w == stemmed_word:
                row[i] = 1

    # return row pattern
    return numpy.array(row)


def chat():
    print("Hi, I am Chatbot, ask me questions about chatbots's, type bye to exit:")
    while True:

        # enter query
        query = input("> ")
        if query.lower() == "bye":
            print("Have a nice day")
            break

        # get neural network reponse for query
        response = get_response(query, stemmed_words)
```

```
    # do prediction
    results = model.predict([response])

    # choose largest result
    results_index = numpy.argmax(results)

    # get category for index
    tag = sorted_categories[results_index]
    print(tag)

    # match category
    responses = []
    for i,category in enumerate(categories):
       if category == tag:
          responses.append(answers[i])

    # pick random response
    print(random.choice(responses))

chat()
```

Here is the complete code for AI Chatbot:

```
"""
AIchatbotex.py
"""

import numpy
import tflearn
#import tensorflow
from tensorflow.python.framework import ops
#ops.reset_default_graph()
import random

# needed to suppress warnings
import warnings
warnings.filterwarnings('ignore')


# download nltk modules
# nltk.download() # for nltk downloading packages (not needed)
```

```
import nltk
nltk.download('punkt') # run once only
#nltk.download('wordnet') # run once only


# get list of stop words
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')

# get punctuation string
import string
punct_dict = dict((ord(punct), None) for punct in string.punctuation)

# make stemmer
import nltk
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

# store info
categories = []
questions = []
answers = []

# open file
f=open("aichatbot.txt","r")

# read lines from file
line = f.readline().strip()
while(line):

   categories.append(line)
   line = f.readline().lower().strip()
   questions.append(line)
   line = f.readline().lower().strip()
   answers.append(line)
   line = f.readline().strip()

# print out data
print("categories:")
print(categories)
print("questions:")
print(questions)
```

```
print("answers:")
print(answers)


# make list of words, sentence words and sentence categories
word_tokens_stop = []
questions_tokenized_stopped = []

# for each question
for i,question in enumerate(questions):
   # remove punctuation
   question=question.translate(punct_dict)
   # word tokenize question
   tokens = nltk.word_tokenize(question)
   # remove stop words
   tokens_stop = [w for w in tokens if not w in stop_words]
   word_tokens_stop.extend(tokens_stop)
   questions_tokenized_stopped.append(tokens_stop)

# print out data
print("word tokens stop:")
print(word_tokens_stop)
print("sentence_tokenized:")
print(questions_tokenized_stopped)


# stem word tokens
stemmed_words = [stemmer.stem(w) for w in word_tokens_stop]
stemmed_words = sorted(list(set(stemmed_words)))

# sort categories
sorted_categories = sorted(list(set(categories)))

training = []
output = []

# map questions to words
for i, question in enumerate(questions_tokenized_stopped):

   training_row = []

   # stem tokens
   stemmed_question = [stemmer.stem(token) for token in question]
```

```
    # check foe matched word
    for w in stemmed_words:
        if w in stemmed_question:
            training_row.append(1)
        else:
            training_row.append(0)

    # map question to category
    output_row = [0] * len(sorted_categories)
    output_row[sorted_categories.index(categories[i])] = 1

    training.append(training_row)
    output.append(output_row)


# neural net input and outputs
training = numpy.array(training)
output = numpy.array(output)

#tensorflow.reset_default_graph()
ops.reset_default_graph()


# make neural net
net = tflearn.input_data(shape=[None, len(training[0])])
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, 8)
net = tflearn.fully_connected(net, len(output[0]), activation="softmax")
net = tflearn.regression(net)

# make neural network model
model = tflearn.DNN(net)


# train neural network
model.fit(training, output, n_epoch=1000, batch_size=8, show_metric=True)


# get response from neural network
def get_response(query, words):

    # empty row
    row = [0] * len(words)
```

```
    # to lower case
    query = query.lower()

    # remove punctuation
    query=query.translate(punct_dict)

    # word tokenize question
    tokens = nltk.word_tokenize(query)

    #remove stop words
    tokens_stop = [w for w in tokens if not w in stop_words]

    # stem words
    stemmed_tokens = [stemmer.stem(word) for word in tokens_stop]

    # map query to question words
    for stemmed_word in stemmed_tokens:
        for i, w in enumerate(words):
            if w == stemmed_word:
                row[i] = 1

    # return row pattern
    return numpy.array(row)


def chat():
    print("Hi, I am Chatbot, ask me questions about chatbots's, type bye to exit:")
    while True:

        # enter query
        query = input("> ")
        if query.lower() == "bye":
            print("Have a nice day")
            break

        # get neural network reponse for query
        response = get_response(query, stemmed_words)

        # do prediction
        results = model.predict([response])

        # choose largest result
```

```
    results_index = numpy.argmax(results)

    # get caregory for index
    tag = sorted_categories[results_index]
    print(tag)


    # match category
    responses = []
    for i,category in enumerate(categories):
       if category == tag:
          responses.append(answers[i])

    # pick random response
    print(random.choice(responses))

chat()
```

to do:

Type in or copy and pastein the above code and run it. Try out sone queries.

You should get something like this:

Hi, I am Chatbot, ask me questions about chatbots's, type bye to exit:

> Who were the early charbot

Development

early chatbots are eliza (1966) and parry (1972) and later a.l.i.c.e., jabberwacky and d.u.d.e

> How does Eliza work?

Limitations

efficiency highly depends on language processing , require a large amount of conversational data to train

> What is a chat bot?

Applications

messaging apps, company internal platforms, customer service, and healthcare

> What are applications for chatbots

Applications

messaging apps, company internal platforms, customer service, and healthcare

> What are the limitations of Chatbots?

Limitations

efficiency highly depends on language processing , require a large amount of conversational data to train

> bye

Have a nice day

NaturalLanguageProcessing Question 2

Replace the random choice instead use a `TfidfVector and` then calculate the cosine similarity and choose the best response.
Put your python code in a file called natural_language2.py

END