application

Omkar Deshpande Follow
May 5, 2017 · 3 min read

Word Sense Disambiguation (WSD), has been a trending area of research in Natural Language Processing and Machine Learning. WSD is basically solution to the ambiguity which arises due to different meaning of words in different context.

For example, consider the two sentences.

"The ***bank*** will not be accepting cash on Saturdays. "

"The river overflowed the ***bank.***"

The word ***bank*** in the first sentence refers to the commercial (finance) banks, while in second sentence, it refers to the river bank. The ambiguity that arises due to this, is tough for a machine to detect and resolve. Detection of ambiguity is the first issue and resolving it and displaying the correct output is the second issue. Here, the code presented is for solving the second issue. Feel free to contribute to it.

. . .

Show me the code…

The imports we will be using are,

```
1  import nltk
2  import codecs
3  from nltk.tokenize import PunktSentenceTokenizer,sent_t
4  from nltk.corpus import stopwords, wordnet
```

> *Natural Language Toolkit (NLTK) library, is free, open source tool developed by Princeton University. This library does the core functioning for our application. It provides training data sets, Wordnet corpus, various tokenizers , lemmatizers, stemmers and taggers.*

```
def simpleFilter(sentence):

        filtered_sent = []
        lemmatizer = WordNetLemmatizer()
        stop_words = set(stopwords.words("english"))
        words = word_tokenize(sentence)

        for w in words:
                if w not in stop_words:
```

The *simpleFilter* function takes the given query/sentence as an input and returns list of tokens which are lemmatized. *Lemmatization* refers to deriving the root word which is morphologically correct. There rises a slight confusion between ***lemmatization*** and ***stemming.*** However, the two words differ in their flavor. *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the ***lemma***.

Stopwords, are the high frequency words in a language which do not contribute much to the topic of the sentence. In English, such words include, 'a' , 'an' , 'the', 'of' , 'to' , etc.. We remove these words and focus on our main subject/topic, to solve ambiguity. The function is applied to the training data sets as well as user input.

```
def simlilarityCheck(word1, word2):

        word1 = word1 + ".n.01"
        word2 = word2 + ".n.01"
        try:
                w1 = wordnet.synset(word1)
                w2 = wordnet.synset(word2)

                return w1.wup_similarity(w2)
```

Next we perform similarity check, function : *similarityCheck*, for the filtered sentence tokens that are returned by the first function. Similarity is checked between the given query/sentence tokens and the training data set tokens. For this, the synonym set is loaded for each

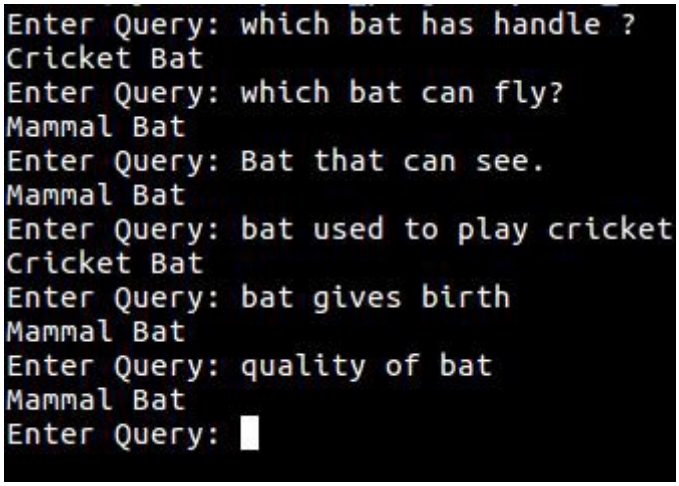
token word from *wordnet* corpus. The depth and closeness of a word is calculated and returned on scale of 0–1 . This is the main data that will resolve the ambiguity. The more data you provide, the more accurate it gets. The normalised similarity between sentences is stored.

synonymsCreator is a simplistic function to store the synonyms of the given input word. This will be used is storing the synonyms of the given data set and query tokens. The synonyms will also be taken into consideration while performing similarity check for the sentences.

```
1
2  # Remove Stop Words . Word Stemming . Return new token
3  def filteredSentence(sentence):
4
5          filtered_sent = []
6          lemmatizer = WordNetLemmatizer()   #lemmatizes
7          ps = PorterStemmer()    #stemmer stems the roo
8
9          stop_words = set(stopwords.words("english"))
10         words = word_tokenize(sentence)
11
12         for w in words:
```

Once the similarity is stored, we apply the next level filter, function: *filteredSentence* , to apply lemmatization over stemmed tokens and again removing stop words. In the filtered sentence list, we now store the token word along with its synonyms for more precised matching / similarity check. Next, we put all these together.

What done here, is, the application has been fed with two data set files, first ***cricketbat.txt*** , which contains few sentences referring to bat used in cricket sport, and second, ***vampirebat.txt***, which contains few sentences referring to the mammal bird bat. *sent1* stores the lowered case string data from the vampirebat.txt file and sent2 does for cricketbat.txt, sent3 stores user query. Next, the sentences are filtered and similarity is checked using the functions explained above. The comparison is normalised, and output is given accordingly whether the query refers to ***Cricket bat*** or ***Mammal bat*** **.**



The program gives quiet accurate answers. The only thing it cannot handle are the negation sentences, like "*which bat is* ***not*** *used to play cricket*" , "*which bat does* ***not*** *fly" , etc.* Feel free to contribute modules to it HERE.

The data set and code has been included in the repository.