**Converting words to their base forms using stemming**

Working with text means working with a lot of variation. We must deal with different forms of the same word and enable the computer to understand that these different words have the same base form. For example, the word sing can appear in many forms, such as *singer*, *singing*, *song, sung*, and so on. This set of words share similar meanings. This process is known as stemming. Stemming is a way of producing morphological variants of a root/base word. Humans can easily identify these base forms and derive context.

When analyzing text, it's useful to extract these base forms. Doing so enables the extraction of useful statistics derived from the input text. Stemming is one way to achieve this. The goal of a stemmer is to reduce words from their different forms into a common base form. It is basically a heuristic process that cuts off the ends of words to extract their base forms. Let's see how to do it using NLTK.

**Create a new Python file and import the following packages:**

from nltk.stem.porter import PorterStemmer

from nltk.stem.lancaster import LancasterStemmer

from nltk.stem.snowball import SnowballStemmer

**Define some input words:**

input\_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',

'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']

Create objects for the **Porter**, **Lancaster**, and **Snowball** stemmers:

***# Create various stemmer objects***

porter = PorterStemmer()

lancaster = LancasterStemmer()

snowball = SnowballStemmer('english')

Create a list of names for table display and format the output text accordingly:

***# Create a list of stemmer names for display***

stemmer\_names = ['PORTER', 'LANCASTER', 'SNOWBALL']

formatted\_text = '{:>16}' \* (len(stemmer\_names) + 1)

print('\n', formatted\_text.format('INPUT WORD', \*stemmer\_names),

'\n', '='\*68)

**Iterate through the words and stem them using the three stemmers**:

# Stem each word and display the output

for word in input\_words:

output = [word, porter.stem(word),

lancaster.stem(word), snowball.stem(word)]

print(formatted\_text.format(\*output))

The full code is given in the file stemmer.py. If you run the code, you will get the following output:



Figure 2: Stemmer output

Let's talk about the three stemming algorithms that are being used here. All of them basically try to achieve the same goal. The difference between them is the level of strictness that's used to arrive at the base form.

The Porter stemmer is the least strict, and Lancaster is the strictest. If you closely observe the outputs, you will notice the differences. Stemmers behave differently when it comes to words such as possibly or provision. The stemmed outputs obtained from the Lancaster stemmer are a bit obfuscated because it reduces the words a lot. At the same time, the algorithm is fast. A good rule of thumb is to use the Snowball stemmer because it's a good trade-off between speed and strictness.

**Converting words to their base forms using lemmatization**

**Lemmatization** is another method of reducing words to their base forms. In the previous section, we saw that some of the base forms that were obtained from those stemmers didn't make sense. Lemmatization is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is like stemming, but it brings context to the words. So, it links words with similar meanings to one word. For example, all three stemmers said that the base form of *calves* is *calv*, which is not a real word. Lemmatization takes a more structured approach to solve this problem. Here are some more examples of lemmatization:

* rocks : rock
* corpora : corpus
* worse : bad

The lemmatization process uses the lexical and morphological analysis of words. It obtains the base forms by removing the inflectional word endings such as *ing* or *ed*. This base form of any word is known as the lemma. If you lemmatize the word *calves*, you should get *calf* as the output. One thing to note is that the output depends on whether the word is a verb or a noun. Let's look at how to do this with NLTK.

**Create a new Python file and import the following packages:**

from nltk.stem import WordNetLemmatizer

**Define some input words. We will be using the same set of words that we used in the previous section so that we can compare the outputs:**

input\_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',

'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']

Create a lemmatizer object:

*# Create lemmatizer object*

lemmatizer = WordNetLemmatizer()

**Create a list of lemmatizer names for the table display and format the text accordingly**:

*# Create a list of lemmatizer names for display*

lemmatizer\_names = ['NOUN LEMMATIZER', 'VERB LEMMATIZER']

formatted\_text = '{:>24}' \* (len(lemmatizer\_names) + 1)

print('\n', formatted\_text.format('INPUT WORD', \*lemmatizer\_names),

'\n', '='\*75)

**Iterate through the words and lemmatize the words using noun and verb lemmatizers:**

*# Lemmatize each word and display the output*

for word in input\_words:

output = [word, lemmatizer.lemmatize(word, pos='n'),

lemmatizer.lemmatize(word, pos='v')]

print(formatted\_text.format(\*output))

The full code is given in the file lemmatizer.py. If you run the code, you will get the following output:

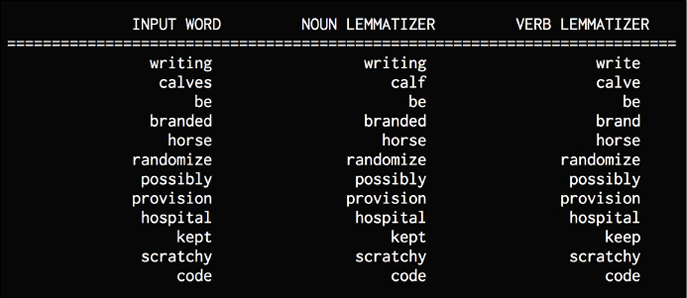


Figure 3: Lemmatizer output

We can see that the noun lemmatizer works differently than the verb lemmatizer when it comes to words such as writing or calves. If you compare these outputs to stemmer outputs, you will see that there are differences too. The lemmatizer outputs are all meaningful, whereas stemmer outputs may or may not be meaningful.