NLP Course 2 Week 1 Lesson : Building The Model - Lecture Exercise 02

Estimated Time: 20 minutes

Candidates from String Edits

Create a list of candidate strings by applying an edit operation

Imports and Data

In [1]:

```
# data
word = 'dearz' # □
```

Splits

Find all the ways you can split a word into 2 parts !

In [2]:

```
# splits with a loop
splits_a = []
for i in range(len(word)+1):
    splits_a.append([word[:i],word[i:]])

for i in splits_a:
    print(i)
```

```
['', 'dearz']
['d', 'earz']
['de', 'arz']
['dea', 'rz']
['dear', 'z']
['dearz', '']
```

In [3]:

```
# same splits, done using a list comprehension
splits_b = [(word[:i], word[i:]) for i in range(len(word) + 1)]

for i in splits_b:
    print(i)
```

```
('', 'dearz')
('d', 'earz')
('de', 'arz')
('dea', 'rz')
('dear', 'z')
('dearz', '')
```

Delete Edit

Delete a letter from each string in the `splits` list.
What this does is effectivly delete each possible letter from the original word being edited.

In [4]:

```
# deletes with a loop
splits = splits_a
deletes = []
```

```
print('word : ', word)
for L,R in splits:
    if R:
        print(L + R[1:], ' <-- delete ', R[0])
```

```
word :  dearz
earz  <-- delete  d
darz  <-- delete  e
derz  <-- delete  a
deaz  <-- delete  r
dear  <-- delete  z
```

It's worth taking a closer look at how this is excecuting a 'delete'.
Taking the first item from the `splits` list :

In [5]:

```
# breaking it down
print('word : ', word)
one_split = splits[0]
print('first item from the splits list : ', one_split)
L = one_split[0]
R = one_split[1]
print('L : ', L)
print('R : ', R)
print('*** now implicit delete by excluding the leading letter ***')
print('L + R[1:] : ',L + R[1:], ' <-- delete ', R[0])
```

```
word :  dearz
first item from the splits list :  ['', 'dearz']
L :
R :  dearz
*** now implicit delete by excluding the leading letter ***
L + R[1:] :  earz  <-- delete  d
```

So the end result transforms **'dearz'** to **'earz'** by deleting the first character.
And you use a **loop** (code block above) or a **list comprehension** (code block below) to do this for the entire `splits` list.

In [6]:

```
# deletes with a list comprehension
splits = splits_a
deletes = [L + R[1:] for L, R in splits if R]

print(deletes)
print('*** which is the same as ***')
for i in deletes:
    print(i)
```

```
['earz', 'darz', 'derz', 'deaz', 'dear']
*** which is the same as ***
earz
darz
derz
deaz
dear
```

Ungraded Exercise

You now have a list of ***candidate strings*** created after performing a **delete** edit.
Next step will be to filter this list for ***candidate words*** found in a vocabulary.
Given the example vocab below, can you think of a way to create a list of candidate words ?
Remember, you already have a list of candidate strings, some of which are certainly not actual words you might find in your vocabulary !

So from the above list **earz, darz, derz, deaz, dear**.

You're really only interested in **dear**.

In [13]:

```
vocab = ['dean','deer','dear','fries','and','coke']
edits = list(deletes)

print('vocab : ', vocab)
print('edits : ', edits)

candidates=[]

### START CODE HERE ###
candidates = set(vocab).intersection(set(edits))
### END CODE HERE ###

print('candidate words : ', candidates)
```

```
vocab :  ['dean', 'deer', 'dear', 'fries', 'and', 'coke']
edits :  ['earz', 'darz', 'derz', 'deaz', 'dear']
candidate words :  {'dear'}
```

Expected Outcome:

vocab : ['dean', 'deer', 'dear', 'fries', 'and', 'coke']
edits : ['earz', 'darz', 'derz', 'deaz', 'dear']
candidate words : {'dear'}

Summary

You've unpacked an integral part of the assignment by breaking down **splits** and **edits**, specifically looking at **deletes** here.
Implementation of the other edit types (insert, replace, switch) follows a similar methodology and should now feel somewhat familiar when you see them.
This bit of the code isn't as intuitive as other sections, so well done!
You should now feel confident facing some of the more technical parts of the assignment at the end of the week.