AIR Question 2

April 9, 2020

1 Question 2

Before tackling the problem head on, I wanted to first review the definition given on wikipedia and then make adjustments

[1]:
```
def hamming_distance(string1, string2):
    dist_counter = 0
    for n in range(len(string1)):
        if string1[n] != string2[n]:
            dist_counter += 1
    return dist_counter
```

[2]:
```
# just trying out two of the test cases from the document
print(hamming_distance('make', 'mage'))
print(hamming_distance('MaiSY', 'MaiZy'))
```

```
1
2
```

1.1 Modification 1

The first modification to be made is to add another point for capitilization, unless it occurs in the first position

[3]:
```
def hamming_distance1(s1, s2):
    distance = 0
    for n in range(len(s1)):
        if n == 0: # applying a different rule if we are looking at the first␣
↪position
            if s1[n].lower() != s2[n].lower():
                distance +=1
        else:
            if s1[n].lower() != s2[n].lower(): # modification of original rule
                distance +=1

            # The following could be on one line in an or statement, but I␣
↪visually prefer to see the logic this way

            if s1[n].isupper() and s2[n].islower(): # checking for␣
↪capitilzation and applying excpection
                distance += 0.5
            if s1[n].islower() and s2[n].isupper(): # checking for␣
↪capitilzation and applying excpection
                distance += 0.5

    return distance
```

[4]:
```
# testing on the exaplmes
print(hamming_distance1('Kitten','kitten'))
print(hamming_distance1('kitten','kiTten'))
print(hamming_distance1('Puppy', 'POppy'))
```

```
0
0.5
1.5
```

1.2 Modification 2

The second modification considers s and z to be the same letter. The previous rules still apply

[5]:
```
# to make this more dynamic, lets create a list with exceptions that we can add␣
 ↪to or remove later
exception_pairs = [('s','z'),('z','s'),('S','Z'),('Z','S')]
```

[6]:
```
def hamming_distance_Final(s1, s2):
    distance = 0
    for n in range(len(s1)):
        if n == 0:
            if s1[n].lower() != s2[n].lower():
                distance +=1
        else:
            if s1[n].lower() != s2[n].lower():
                if (s1[n],s2[n]) not in exception_pairs: #applying our last␣
 ↪exeception
                    distance += 1
            if s1[n].isupper() and s2[n].islower():
                distance += 0.5
            if s1[n].islower() and s2[n].isupper():
                distance += 0.5

    return distance
```

[7]:
```
# all text cases explored
print(hamming_distance_Final('make', 'Mage'))
print(hamming_distance_Final('MaiSY', 'MaiZy'))
print(hamming_distance_Final('Eagle','Eager'))
print(hamming_distance_Final('Sentences work too','Sentences wAke too'))
```

```
1
0.5
2
3.5
```

1.3 Scoring Given Strings

[8]:
```
print('The score for a) is: '+ str(hamming_distance_Final("data Science","Data␣
 ↪Sciency")))
print('The score for b) is: '+␣
 ↪str(hamming_distance_Final("organizing","orGanising")))
print('The score for c) is: '
      +␣
 ↪str(hamming_distance_Final("AGPRklafsdyweIllIIgEnXuTggzF","AgpRkliFZdiweIllIIgENXUTygSF")))
```

```
The score for a) is: 1
The score for b) is: 0.5
The score for c) is: 8.5
```

1.4 Applications of Hammering Distance

One of the applicatins of Hammering Distance I see right away is for spellchecking or spell formatting certain texts. A Hammering Distance above 0 means that that there is something wrong between two texts or two words being compared during a spellcheck algortithim. RegEx can cut through a document and then it can be coupled with this algorithim to give an overall measure of how mispelled a document is. The first exception we made helps when dealing with proper nouns or the beginning of a sentence. The second exception I see as especially useful for comparing US and UK texts, given that we are ignoring the counting for s and z.