discrete_inference

February 1, 2019

0.1 Spell Checking

Spell checking problem cab be solved with bayesian theory. The problem we need to solve is:

$$argmax_{correct\ word}(P(correct\ word \mid wrong\ word) = \frac{P(wrong\ word \mid correct\ word)P(correct\ word)}{P(wrong\ word)})$$

$P(wrong\ word \mid correct\ word)$ can be measured with edit distance. $P(correct\ word)$ is the overall word frequency. We can download a corpus to obtain the word frequency. The denominator is just a scaling factor.

0.1.1 Obtaining P(correct word)

```
In [1]: import re,collections
```

```
In [47]: def get_words(text): return re.findall('[a-z]+', text.lower())
         words = get_words(open('corpus.txt').read())
         frequency = collections.Counter(words)
```

0.1.2 Obtaining P(wrong word | correct word)

For simplicity, we only consider the wrong spellings 1 and 2 edit distance away from the correct spelling. We consider 4 types of mistake - deletion, transpose, replacement and insertion.

```
In [11]: alphabet = 'abcdefghijklmnopqrstuvwxyz'
         def edits1(word):
             splits = [(word[:i], word[i:]) for i in range(len(word) + 1)]
             #delete one character
             deletes = [a + b[1:] for a, b in splits if b]
             #exchange two consecutive characters
             transposes = [a + b[1] + b[0] + b[2:] for a, b in splits if len(b)>1]
             #change one character to another
             replaces = [a + c + b[1:] for a, b in splits for c in alphabet if b]
             #insert one character
             inserts = [a + c + b for a, b in splits for c in alphabet]
             return set(deletes + transposes + replaces + inserts)
```

The first 10 misspellings of 'word':

```
In [51]: list(edits1('word'))[:10]

Out[51]: ['wcord',
          'world',
          'wurd',
          'wxrd',
          'wmord',
          'wogrd',
          'wori',
          'wordm',
          'wod',
          'ward']
```

We can notice that some are not real English words. We can consider only those in the frequency table and discard the rest of them.

```
In [60]: def known(words): return set(w for w in words if w in frequency)
         known(edits1('word'))

Out[60]: {'cord',
          'ford',
          'lord',
          'ord',
          'sword',
          'ward',
          'wood',
          'word',
          'words',
          'wordy',
          'wore',
          'work',
          'world',
          'worm',
          'worn'}
```

For the edits that are 2 edit distance away, we can take those one edit distance away and edit them once more.

```
In [64]: def known_edits2(word):
             return set(e2 for e1 in edits1(word) for e2 in edits1(e1) if e2 in frequency)
         list(known_edits2('word'))[:10]

Out[64]: ['world',
          'god',
          'ward',
          'wards',
          'worn',
          'wound',
          'tore',
```

```
'cord',
'words',
'woof']
```

0.1.3 Test

Now we have all we need. The candidates for spell correction are chose with a priority: - The original word, if it is known - The list of known words one edit distance away - The list of known words two edit distance away - The original word

For candidates of the same priority, we choose the one with highest word frequency as the final result.

```
In [99]: def candidates(word):
             return known([word]) or known(edits1(word)) or known_edits2(word) or word

         def correct(word):
             return max(candidates(word), key=lambda x:frequency[x] if not frequency[x] is None

In [103]: candidates('aol')

Out[103]: {'al', 'all', 'awl', 'col', 'gaol', 'sol', 'vol'}

In [104]: correct('aol')

Out[104]: 'all'

In [106]: candidates('EECS')

Out[106]: 'EECS'

In [107]: candidates('probablity')

Out[107]: {'probability'}

In [108]: correct('probablity')

Out[108]: 'probability'

In [109]: candidates('halp')

Out[109]: {'hale', 'half', 'hall', 'halo', 'halt', 'harp', 'hasp', 'help'}

In [110]: correct('halp')

Out[110]: 'half'

In [120]: correct_sentence = lambda sentence:' '.join(map(correct,sentence[:-1].split(' ')))
          correct_sentence('I need somee halp!')

Out[120]: 'a need some half'

In [129]: correct_sentence('He hates the asignments.')

Out[129]: 'he rates the assignment'
```

0.1.4 Conclusion

We can see our model can correct spellings, but it's rather naive. It doesn't take context and inflection into consideration.