EDAN20 Language Technology
-Lab 0.

Nils Romanus

September 2022

1 The Norvig Spell Checker

Using basic python coupled with the regex library Norvig has been able to construct a basic yet functioning spell-checker. This report gives an outline of the different functions in the script and how they form the spell-checking program.

2 The *words* function

```
def words(text): return re.findall(r'\w+', text.lower())
```

Listing 1: The words function

The words function finds all words in the input text. This is done by using the regex findall function using the \w pattern. The \w pattern works for all languages using the python regex module as opposed to how it works in Perl. The function returns a list of strings containing the words of the text. The findall call uses no flags.

```
WORDS = Counter(words(open('big.txt').read()))
```

Listing 2: Calling the words function

The words function is subsequently called with the text in the "big.txt" file as input. The "big.txt" file hence constitutes the corpus used in the script. Using the *Counter* module from the *Collections* library the list of words that are output from the *words* function is subsequently transformed into a dictionary with the words as keys and the number of occurrences in the corpus as corresponding values.

3 The *edits1*- and *edits2* functions

```
def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word)
    + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len
    (R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R
    for c in letters]
    inserts    = [L + c + R               for L, R in splits for c
    in letters]
    return set(deletes + transposes + replaces + inserts)
```

Listing 3: The edits1 functions

The *edits1* finds all possible results from a simple edit of the word *word*. A simple edit implies adding or removing a letter, swapping adjacent letters, or

changing one letter to another. These are generated from basic list comprehension operations. The results are subsequently concatenated and returned as a set of simple edits thus removing the edits that are identical.

```
def edits2(word):
    "All edits that are two edits away from ‘word‘."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

Listing 4: The edits1 functions

The *edits2* function calls on the *edits1* function to generate the set of corrections that require two simple edits using a nested list (set) comprehension.

4 The *canditates* function

```
def known(words):
    "The subset of ‘words‘ that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)
```

Listing 5: The edits1 function

The known function returns the subset of words that appear in the corpus dictionary.

```
def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(
    word)) or [word])
```

Listing 6: The edits2 function

Using the *known-*, *edits1-* and *edits2* functions the *candidates* function returns a tuple of the possible candidates for the, correctly- or misspelled, word "word".

5 The *P* function

```
def P(word, N=sum(WORDS.values())):
    "Probability of ‘word‘."
    return WORDS[word] / N
```

Listing 7: The P function

The P function estimates the probability of the word "word" by computing the relative frequency of the word in the corpus.

6 The *correction* function

```
def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)
```

Listing 8: The correction function

The *correction* function is the one that is actually applied when using the spell-checker. It works by firstly generating all the possible candidates using the *candidates* function, i.e the words that are known, the known first-order candidates, and the known second-order candidates. If the word itself and its first- and second-order edits are not known, i.e if these constitute an empty set, the input word is used. The max function is subsequently applied to this set using the P function as the *key* argument. The *key* argument in the standard python *max* takes a function that specifies how the elements in the set are compared. Hence the max function will return the argument in the set *candidates(word)* that results in the largest probability. This results in the candidate with the largest probability beeing returned, i.e the most likely word that the user wanted to write.