UnSCRABBLEing Mathematics

Samuel Ridgway

December 10, 2015

1 Introduction

Mathematicians and Linguists are often at odds with one another. Both scoff at the other's academic achievements, safe in the knowledge that their area of study is the more rewarding and applicable. However there is a point in every person's life when these two schools of thought combine to require rudimentary knowledge of both vocabulary and arithmetic. I am of course referring to the annual game of Scrabble forced upon us by senile relatives during the holiday period. During the course of this assignment I will be exploring a simple way to code your way out of yet another humiliating defeat at the hands of Aunt Mabel.

2 Coding Scrabble

2.1 The problem

Humour aside, the simplest way to approach this task is to input your seven letters, then use code to generate all possible combinations and check them against a dictionary. The problem with this approach is that with the English dictionary containing at least a quarter of a million distinct words [1] checking every possible combination of letters to find one that is valid will take a long time (relative to computer functionality). Therefore I have explored a different approach which will hopefully take the computer a shorter amount of time to complete.

2.2 The approach

In planning the code for this venture, I had to take a few assumptions and liberties:

- Firstly, I focused on creating 7 letter words which would use all the letters in your 'hand' to achieve the bonus 50 points. This could also be used in conjunction with any available letter on the board (but would not guarantee a 50 point bonus).
- Also, I haven't created any code to score the words. This is because I haven't taken into account what the layout of the board (i.e. triple or double bonuses) which could mean 2 to 6 letter words could score higher or be more strategic. Generally 7 letters is the best combination without knowing the board.

2.3 The solution

The first thing I did was to download a textfile with the complete list of words used in most Scrabble tournaments across the globe, this list is called SOWPODS [2].

```
textfile = open('Scrabble Dictionary.txt', 'r')
wordlist = textfile.read()  # Read the words into a list
words = wordlist.split('\n')  # Split the list into row form
```

Now we need a function to generate valid words. I tried to keep things simple while speeding up the process.

```
def wordgenerator():
    """
    A function to determine whether the letters in the list can
    form a valid 7 letter word
    """
    valid1 = []  # Set up a starting list
    for row in words:  # Check each row for the first letter
        if letters[0] in row:
            valid1.append(row)  # Append each valid word
    """
    Continue for each letter, creating smaller and smaller lists
    """
    valid2 = []
    for string in valid1:
        if letters[1] in string:
            valid2.append(string)
    valid3 = []
    for string in valid2:
        if letters[2] in string:
            valid3.append(string)
    valid4 = []
    for string in valid3:
        if letters[3] in string:
            valid4.append(string)
    valid5 = []
    for string in valid4:
        if letters[4] in string:
            valid5.append(string)
    valid6 = []
    for string in valid5:
        if letters[5] in string:
            valid6.append(string)
    valid7 = []
    for string in valid6:
        if letters[6] in string:
            valid7.append(string)
    print valid7  # This lists all the valid 7+ letter words
```

When I ran this in atom with the sample letters ['x', 'o, 'j', 'k', 'u', 'b', 'e'] the time taken to finish the function was 0.244s, which I believe is considerably faster than if I had used the 'simplest approach' as outlined above. The solutions were 'jukebox' and 'jukeboxes'.

The reason I believe this is one of the best approaches is because unlike the 'simplest approach' the list of valid words gets smaller as you go through the function, meaning that each consecutive letter would be checked with fewer and fewer 'strings'. One way to utilise this to it's fullest efficiency is to input the highest scoring letter (often the least frequent letter) into the input method as shown below.

```
import random
letters = input("Write your seven letters here, in form ['a', 'b'
,...] ->")
"""
This simple code will allow the user to input their letters into
a list form to be used in the rest of the code
"""
```

3 Conclusion

So we can see that with careful planning, what started out as a slow, clunky function, can be streamlined into a much faster, more efficient method of generating relative-crushing words.

It would be interesting to see what alterations we could make to the code if we were to expand the premise to include a playing board. But as this would likely take up more than 3 pages of LaTeX, I think we will leave that to another time. Hopefully what I have done so far will bring much more joy to the game, just don't tell your opponents how you've become a linguistic genius while away studying a maths degree!

References

[1] Oxford University Press. How many words are there in the english language?, 2015. [Online; accessed: 08-12-2015].

[2] Wikipedia. Sowpods — wikipedia, the free encyclopedia, 2014. [Online; accessed: 8-12-2015].