List of Concepts

April 12, 2020

0.1 *Variable assignment and algebraic manipulation*

This code assigns the values 5 and 2 to a and b respectively, then prints the value of 5 mod 2.

[1]:
```
a = 5
b = 2
print(a % b)
```

```
1
```

0.2 *Single conditional statements*

This code checks the sign of a random integer between -5 and 10.

[7]:
```
import random

n = random.randint(-5, 10)
if n < 0: #checks if n is negative
    print("n = ", n, ". n is negative.")
elif n > 0: #checks if n is positive
    print("n = ", n, ". n is positive.")
else: #if n is neither positive nor negative, it must be zero
    print("n is zero.")
```

```
n =  7 . n is positive.
```

0.3 *Repeated conditional statements*

This code flips a coin until it gets heads, and checks how many tails it gets before this.

[42]:
```
import random

tails = 0
heads_achieved = False

while heads_achieved == False: #repeat until a head is achieved
    if random.randint(0, 1) == 1: #where 1 is a head
        print("Heads!")
        heads_achieved = True #stops the loop
    else: #else, it is a tails; repeat loop
```

```
        print("Tails!")
        tails += 1
print(tails)
```

```
Tails!
Tails!
Tails!
Tails!
Tails!
Tails!
Heads!
6
```

0.4 *Functions and lists*

This function creates a list of all even numbers smaller than a given number.

[29]:
```
def evens(n):
    '''Creates a list of all even numbers smaller than n'''
    results = [] #creates an empty list in which to put the numbers
    for i in range (n):
        if n > 2 * i: #where 2 * i is the even sequence 2,4,6...
            results.append(2 * i)
    return results
```

[30]:
```
evens(22)
```

[30]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

0.5 *Iteration*

This code checks how many vowels are in a string.

[3]:
```
word = "Dublin, Ireland"
letters = list(word.lower())
vowels = 0

for n in range (len(word)): #loops for as many letters as the list contains
    if letters[n] in ['a', 'e', 'i', 'o', 'u']: #if the letter being checked is␣
 ↪a vowel
        vowels += 1
print(vowels)
```

```
5
```

0.6 *Recursion*

This function finds the n'th triangular number.

[43]:
```
def triangular(n):
    '''Finds the nth triangular number by recursion.'''

    if n <= 1:
        return 1
    else:
        return n + triangular(n-1)
```

[44]:
```
triangular(5)
```

[44]:
```
15
```

0.7 *Object-oriented programming*

This class represents vectors in 3-dimensional space.

[64]:
```
class Vector:
    '''A class for vectors in 3-dimensional space.'''

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __add__(self, other): #adds two vectors together
        return Vector(self.x + other.x, self.y + other.y, self.z + other.z)

    def dot_product(self, other): #gives the scalar (dot) product of two vectors
        return (self.x * other.x + self.y * other.y + self.z * other.z)

    def cross_product(self, other): #gives the vector (cross) product of two␣
↪vectors
        return Vector(self.y * other.z - self.z * other.y, self.z * other.x -␣
↪self.x * other.z, self.x * other.y - self.y * other.x)

    def __repr__(self): #represents vectors as a string, rather than an object
        return str([self.x, self.y, self.z])
```

[65]:
```
A = Vector(2, 3, 4)
B = Vector(1, 5, 1)
```

[66]:
```
A + B
```

[66]:
```
[3, 8, 5]
```

[67]:
```
A.dot_product(B)
```

[67]: 21

[68]:
```
A.cross_product(B)
```

[68]: [-17, 2, 7]

[70]:
```
#this should be the inverse of A x B
B.cross_product(A)
```

[70]: [17, -2, -7]

Countdown: The Numbers Behind Letters

Alex Room

April 15, 2020

Channel 4's TV show Countdown consists of two rounds; a numbers round, where players take turns to choose a set of 6 numbers (from their choice number of 'large' and 'small' numbers), then must use them (but only once each) with basic arithmetic to equal a randomly generated number. The letters round, meanwhile, has players choose random vowels and consonants to form a board of 9 letters, and then must find the longest word they can create from these letters. The former has been extensively studied; even so far as solved [1], whereas the letters round is much less so; the closest similar field is that of Scrabble, a board game where players similarly create words from letters to connect, crossword style, across a board - this game has had decades of algorithms studied to create AI players [2], and has been applied to many fields, as far as economics (Polak [3]). This paper will aim to use generating Countdown boards to garner information about the similarity between languages.

In this paper, we simulate rounds of Countdown's letters round in order to statistically analyse the largest words possible; in particular, which number of vowels is ideal to have room for the largest words (and thus the most points)?

We begin by turning a dictionary file into a set. For this paper, I used the SOWPODS scrabble words dictionary; this is simply as this dictionary doesn't contain proper nouns, pre/suffixes, and other words not allowed in Countdown.

```
with open('sowpods.txt', 'r') as Words:
    string = Words.read()
    sowpods = (string.split('\n'))
    sowpods.remove('')
    sowpods = {str(i) for i in sowpods if len(i) <= 9}
#converts the SOWPODS scrabble dictionary into a set
```

To check this dictionary against our board, we turn the board into a power set[1]; that is, create a set of every possible combination of letters (word or not). Here we sort them from longest to shortest - we are finding the longest word possible on each board, so there's no use checking 36 ($\binom{9}{2}$) 2-letter sets if there are eligible 7-letter sets. This allows us to compare these strings created from the board against those in the dictionary in the form of tuples.

```
def powerset(iterable):
    '''powerset([1,2,3]) -->  (1,2,3) (1,2) (1,3) (2,3) (1,) (2,) (3,) ()'''
    s = list(iterable)
    return itertools.chain.from_iterable(itertools.combinations(s, r) for r in range(len(s)+1, 1, -1))
```

[1] Apart from reversing the order of the sets (to be largest subset first, instead of smallest first), this is identical to the power set function in https://docs.python.org/3/library/itertools.html#itertools-recipes

We then can use this dictionary and definition to create our solve_board function, which creates a countdown board with a given number of vowels and finds the longest word that can be created from it. The variables 'vowels' and 'consonants' are the obvious strings.

```
#turning the dictionary into a set of split letters, each word as a tuple
dictionary = set([tuple(sorted(word)) for word in sowpods])

def solve_board(vowel_number,
                display_board=False,
                dictionary=dictionary,
                vowels=vowels,
                consonants=consonants):
    '''creates a countdown board of 9 letters and returns the largest word present'''
    board = []
    length = [0]

    #randomly generates the board
    board = list(np.random.choice(list(vowels), vowel_number, replace=True))
    board += list(np.random.choice(list(consonants), 9 - vowel_number, replace=True))
    if display_board is True:
        print(board)

    #checks for words that can be created using the board letters,
    #returns the maximum word length possible
    sorted_board = tuple(sorted(board))
    for subset in powerset(sorted_board):
        if subset in dictionary:
            return subset
    return ([])
```

Finally, this function is iterated over a large number of repetitions to find the average longest word for each vowel number,

```
averages = []
for v in range (0, 10):
    highest = []
    for _ in range (0, 50000):
        highest.append(len(solve_board(v)))
    averages.append(sum(highest) / len(highest))
```

then graphed as a curve for each vowel number alongside a fitted quadratic curve (in form $ax^2 + bx + c$, fitted via SciPy) in red.

```
from scipy import optimize

def func(x, a, b, c):
    return a * x ** 2 + b * x + c

popt, pcov = optimize.curve_fit(func, index, averages)
fitted_ys = [func(x, *popt) for x in index]

plt.plot(index, averages, label="Data")
plt.plot(index, fitted_ys, label='fit: a=%5.3f, b=%5.3f, c=%5.3f' % tuple(popt), color="red")
plt.xlabel("Number of vowels")
plt.ylabel("Average word length")
plt.legend()
plt.savefig("english-fitted.png")
```

Generating our graph and fitted curve, we get results thus:

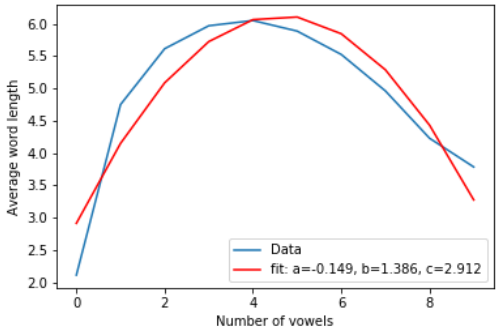
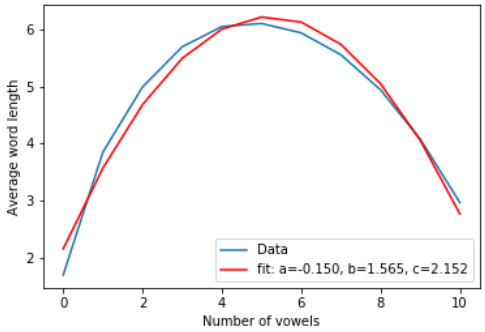


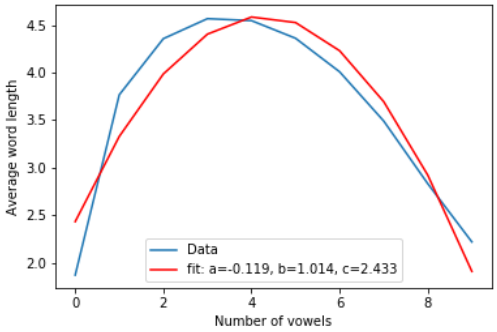
Figure 1: The curve for English words

This would suggest that the ideal number of vowels on the board is 3-4 (4.65 on the parametrised curve), although the consequences of choosing extremely high vowel numbers are less than those of choosing zero or one.

As further analysis, Countdown is inspired by French TV show *Des chiffres et des lettres*, which holds a similar letters round system, but a board of 10 letters is used instead.



(a) The curve for French words



(b) The curve for German words

Similarly simulating French boards of 10 letters, we find a similar curve, with 4-5 vowels as the peak; This is reflected in the parametrised curves, which are incredibly similar for both languages, and have peaks of 5.21 and 4.65 on the curves for French and English respectively. French and English have quite a similar distribution of letters [4], which could explain this. We do this again for German; the shape is again similar, showing similarity in Latin languages, where the length of words and use of specific letters and phonemes stays similar regardless of language due to Zipf's principle of least effort [5]. However, the maximum word length is much lower, peaking at 4.59 vs French's peak of 6.23 on their respective fitted curves; Countdown or an equivalent game has not yet reached German television, and this clash between the game's format and the German language may be at least partially at fault.

References

[1] Simon Colton. Countdown numbers game: Solved, analysed, extended. 2014.

[2] Andrew W. Appel and Guy J. Jacobson. The world's fastest scrabble program. *Commun. ACM*, 31(5):572–578, May 1988.

[3] J. J. Polak. The economics of scrabble. *The American Economic Review*, 45(4):648–652, 1955.

[4] Marie-José Leclercq, Eric Lee, and Macaire Ngomo. Cryptographie et sécurité de l'information. 2007.

[5] George Kingsley Zipf. *Human behavior and the principle of least effort.* Addison-Wesley Press, 1949.