math 1MP term projects

Ben Bolker

09:27 01 April 2015

These projects are **individual** projects; if you discuss anything about them with other students in the class, you should follow the standard rules (don't write anything down while you are engaged with the other students). If you get anything from the web, give references.

Yahtzee

Yahtzee is a dice game. One small subset of the rules specifies that you have three rolls of five (six-sided) dice in which to try to get as many of a kind as possible. You only have to re-roll as many dice as you want, which means that if you have at least two of a kind you can maximize your chances of getting "Yahtzee" (five of a kind) by not re-rolling those dice. Write the following simulation functions:

```
def single_roll(n,fixed=(),sides=6):
    """perform a single roll of a total of n dice (default 6-sided)
       combine the results for the n-len(fixed) rolled dice
       with the fixed values
       return a list containing the result
    """

def new_roll(vals,sides=6):
    """given a list or tuple of die values, pick which to keep
       fixed and which to re-roll (in case of a tie,
       keep the larger values)
       and re-roll them.  For example, if the
       vals is (1,1,2,6,6), you should re-roll (1,1,2)
       Return a tuple of the complete set of values.
       """

def sim_yahtzee(nrolls,n,sides=6):
    """simulate a specified number of dice rolls of n dice
       Return a tuple of die values corresponding to the final roll
    """

def est_yahtzee_prob(nsims,nrolls=3,dice=5,sides=6):
    """estimate the probabilities of Yahtzee (rolling all dice with
       the same probabilities) for a specified number
       of rolls, dice, and sides.  Should return a *single number*
       between 0 and 1 (inclusive) giving the fraction of simulations
       that resulted in 'yahtzee'
```

```
    """
```

Put these functions together to compute the probability of Yahtzee (use 100,000 simulations each) for

- the standard case (3 rolls of 5 6-sided dice)
- 2 to 6 rolls (with the standard 5 6-sided dice)
- 5, 6, or 7-sided dice (in 3 rolls of 5 dice).
- Approximately how many rolls do you need with 5 7-sided dice to get close to the probability in the base case?

n-grams

This is a variant of this project.

An "n-gram" is a set of `n` consecutive words from a document or string; for example, the phrase "now is the time for all good people" contains six 3-grams:

```
## [('the', 'time', 'for'), ('now', 'is', 'the'), ('for', 'all', 'good'), ('all', 'good', 'people'), ('
```

(note that these are not in order; I stored them in a dictionary . . .)

The first part of this project is to tabulate the n-grams in a document.

The structure of the set of n-grams should be a *dictionary* whose keys are tuples of `n-1` words, and whose contents are lists of the next strings found in the document. For example, if we have the string "today is my birthday today is also his birthday" and we want to tabulate the 2-grams, we would get this:

```
## {('my',): ['birthday'], ('his',): ['birthday'], ('today',): ['is', 'is'], ('is',): ['my', 'also'], (
```

```
def get_file(fn):
    """read in an entire file as a single string"""
    f = open(fn)
    r = f.read()
    f.close()
    return(r)

def clean_string(str):
    """remove all punctuation and newlines ('\n') from a string,
       and make it lowercase"""
     ## hint: str.remove(), str.lower()
```

```
def add_ngrams(strList,n=2,cur_dict=dict()):
    """add all n-grams from a string to the current dictionary;
       return the modified dictionary"""

def make_ngram_dict(fn,n=2):
    """make an n-gram dictionary from a file"""
    ### fn is the name of a file in the current directory,
    ###  which you can with open(fn)
    ### The function should iterate through the lines of the file,
    ### using clean_string() and str.split() to separate the
    ### lines into words.  Then
    ### check whether tuples of (n-1) words are present in the dictionary
    ###    if not, add an entry with the next word
    ###    if so, add the word to the list of words following that tuple
```

Once you've done this, you can use a *Markov generator*; **this is no longer part of the assignment**. Given a starting set of `n-1` words, this will pick the next word at random from the appropriate dictionary entry (`numpy.random.choice(x)` will pick a random element from the list `x`) and add it to the list of words. If the last word in the text you used to set up the dictionary is unique, this generator will pick a random value from the dictionary.

```
import numpy.random as npr

def generate_markov_step(cur_list,ngram_dict,n=2):
    """pick a markov step from a dictionary of n-grams"""
    last_words =  tuple(cur_list[-(n-1):])
    if last_words in ngram_dict:
        new_word = npr.choice(ngram_dict[last_words])
    else:
        ngram_list = list(ngram_dict.keys())
        new_word = ngram_list[npr.randint(len(ngram_list))][0]
    cur_list.append(new_word)
    return(new_word)

cur_list = ["which"]  ## set starting (n-1) words for the phrase
for i in range(50):
    generate_markov_step(cur_list,dd)
```