HW1-prob4_dataAnalytics2

March 11, 2015

```
In [3]: import re, sys, random
        from abc import *

        order = 2
        length = 15
        class FileHandler(object):
            __metaclass__ = ABCMeta
            def __init__(self, path):
                self.path = path
            def get_counts(self, order):
                counts = {}
                data = self.get_states()
                for i in xrange(len(data) - order):
                    previous_state = tuple(data[i:i+order])
                    counts.setdefault(previous_state, {})
                    next_state = data[i+order]
                    counts[previous_state].setdefault(next_state, 0)
                    counts[previous_state][next_state] += 1
                return counts
            @abstractmethod
            def get_states(self):
                ’’’returns list of states found in data’’’
            @staticmethod
            @abstractmethod
            def format(states):
                ’’’returns raw format associated with states’’’

        class CharHandler(FileHandler):
            def get_states(self):
                return tuple([x for x in open(self.path).read() if x not in (’\n’, ’\r’)])
            @staticmethod
            def format(states):
                return ’’.join(states)

        class WordHandler(FileHandler):
            def get_states(self):
                words = re.split(r’\s+’, open(self.path).read())
                return tuple([x for x in words if len(x) > 0])
            @staticmethod
            def format(states):
                return ’ ’.join(states)

        class MarkovChain(object):
```

```
    def __init__(self, order, handlers):
        self.order = order
        self.distro = self._get_distro(handlers)
    def _get_distro(self, handlers):
        '''
        returns dictionary
        keys = tuples of previous states
        values = list of tuples (next_state, cdf)
            cdf at index i => cdf of going to any of states in [0,i) = cdf
            {B:1, C:1, A:2} => [(B, 0), (C, 0.25), (A, 0.5)]
        no gurantees on ordering of states
        guarantees cdfs are increasing
        '''

        # get normalized global counts
        global_counts = {}
        for handler in handlers:
            source_counts = handler.get_counts(self.order)
            for previous_states, next_states in source_counts.iteritems():
                total = 1.0 * sum(next_states.values())
                for next_state, count in next_states.iteritems():
                    global_counts.setdefault(previous_states, {})
                    global_counts[previous_states].setdefault(next_state, 0)
                    global_counts[previous_states][next_state] += count / total
        # create cumulative probability distribution
        distro = {}
        for previous_states, next_states in global_counts.iteritems():
            total = 1.0 * sum(next_states.values())
            distro[previous_states] = []
            cdf = 0.0
            for next_state, count in next_states.iteritems():
                distro[previous_states].append((next_state, cdf))
                cdf += count / total
        return distro

    def walk(self, length):
        '''
        returns list of states, using this model's transition probabilities
        when choosing next state based on previous states.
        starts with random states.

        '''
        previous_states = random.choice(list(self.distro.keys()))
        output = list(previous_states)
        while len(output) < length:
            options = self.distro[previous_states]
            next_state = self.choose_next(options)
            output.append(next_state)
            previous_states = previous_states[1:] + (next_state, )
        return output

    def choose_next(self, options):
        '''randomly chooses next state based on probabilities'''
        r = random.random()
        last = None
        for datum, prob in options:
            if r < prob:
                return last
            else:
                last = datum
        return last

Handler = WordHandler

sources = [WordHandler(path) for path in [’/home/vicky/Downloads/janeAustent.txt’]]
mc = MarkovChain(2, sources)
seq = mc.walk(1000)
austen = Handler.format(seq)

sources = [WordHandler(path) for path in [’/home/vicky/Downloads/conanDoyle.txt’]]
mc = MarkovChain(2, sources)
seq = mc.walk(1000)
doyle = Handler.format(seq)
```

In [228]:

In [ ]: