Project 4 (Markov Model) Checklist

Prologue

Project goal: use a Markov chain to create a statistical model of a piece of English text and use the model to generate stylized pseudo-random text and decode noisy messages

Relevant lecture material

- ⇝ Using Data Types
- ⇝ Creating Data Types

Files

- ⇝ `project4.pdf` (project description)
- ⇝ `project4.zip` (starter files for the exercises/problems, `report.txt` file for the project report, and `run_tests.py` file to test your solutions)

Prologue

Besides knowing how to create and use data types, understanding how strings and dictionaries are manipulated is crucial for the project problems

Example on string manipulation:

```
>>> s = ’hello, world!’
>>> i = 2
>>> k = 5
>>> s[i + k]     # the character at i + k
’w’
>>> s[i : i + k] # substring starting at i and ending at i + k - 1
’llo, ’
>>> s[ : i]      # substring starting at 0 and ending at i - 1
’he’
>>> s[i : ]      # substring starting at i and ending at len(s) - 1
’llo, world!’
>>> s[-k : ]     # substring containing the last k characters
’orld!’
>>> s[ : -k]     # substring containing first len(s) - k characters
’hello, w’
```

Prologue

Example on dictionary manipulation:

```
>>> M = {}                       # create an empty dictionary M
>>> M.setdefault('ba', {})       # add key/value pair 'ba'/{} to M
{}                               # since 'ba' didn't exist in M,
                                 # {} (the value just added) is
                                 # returned
>>> M                            # check M
{'ba': {}}
>>> M['ba'].setdefault('n', 0)   # add key/value pair 'n'/0 to the
0                                # dictionary M['ba']; since 'n'
                                 # didn't exist in M['ba'], 0
                                 # (the value just added) is
                                 # returned
>>> M                            # check M
{'ba': {'n': 0}}
>>> M['ba']['n'] += 1            # increment the value
                                 # corresponding to the key 'n'
                                 # in the dictionary M['ba'] by 1
>>> M                            # check M
{'ba': {'n': 1}}
>>> M['ba'].setdefault('n', 42)  # add key/value pair 'n'/42 to the
1                                # dictionary M['ba']; since 'n'
                                 # exists in M['ba'], setdefault()
                                 # simply returns (without
                                 # changing) the corresponding
                                 # value, 1
>>> M                            # check M
{'ba': {'n': 1}}
```

Prologue

```
>>> M.setdefault(’an’, {})        # add key/value pair ’an’/{} to M
{}
>>> M                             # check M
{’ba’: {’n’: 1}, ’an’: {}}
>>> M[’an’].setdefault(’a’, 0)    # add key/value pair ’a’/0 to the
0                                 # dictionary M[’an’]
>>> M                             # check M
{’ba’: {’n’: 1}, ’an’: {’a’: 0}}
>>> M[’an’][’a’] += 1             # increment the value
                                  # corresponding to the key ’a’
                                  # in the dictionary M[’an’] by 1
>>> M                             # check M
{’ba’: {’n’: 1}, ’an’: {’a’: 1}}
>>> M[’an’][’a’] += 1             # increment the value
                                  # corresponding to the key ’a’
                                  # in the dictionary M[’an’] by 1
>>> M                             # check M
{’ba’: {’n’: 1}, ’an’: {’a’: 2}}
>>> list(M.keys())                # get the keys of M
[’ba’, ’an’]
>>> M.values()                    # get the values of M
[{’n’: 1}, {’a’: 2}]
>>> list(M[’ba’].keys())          # get the keys of M[’ba’]
[’n’]
>>> list(M[’ba’].values())        # get the values of M[’ba’]
[1]
>>> list(M[’an’].keys())          # get the keys of M[’an’]
[’a’]
>>> list(M[’an’].values())        # get the values of M[’an’]
[2]
```

Exercises

Exercise 1. (*Password Checker*) Implement the function `is_valid()` in `password_checker.py` that returns `True` if the given password string meets the following specifications, and `False` otherwise:

- At least eight characters long
- Contains at least one digit (0-9)
- Contains at least one uppercase letter
- Contains at least one lowercase letter
- Contains at least one character that is neither a letter nor a number

```
$ python3 password_checker.py Abcde1fg
False
$ python3 password_checker.py Abcde1@g
True
```

Hint: use the `str` methods `isdigit()`, `isupper()`, `islower()`, and `isalnum()`.

Exercises

password_checker.py

```python
import stdio
import sys


# Return True if pwd is a valid password and False otherwise.
def is_valid(pwd):
    check1 = False # length check
    check2 = False # digit check
    check3 = False # upper case check
    check4 = False # lower case check
    check5 = False # alphanumeric check

    # Perform length check on pwd.
    check1 = len(pwd) >= 8

    # Iterate over characters c of pwd.
    for ... in ...:
        # Perform digit check on c.
        if ...:
            ...
        # Perform upper case check on c.
        elif ...:
            ...
        # Perform lower case check on c.
        elif ...:
            ...
        # Perform alphanumeric check on c.
        elif ...:
            ...

    # Return True if all checks are True and False otherwise.
    ...


# Test client [DO NOT EDIT].
def _main():
    pwd = sys.argv[1]
    stdio.writeln(is_valid(pwd))
```

Exercises

password_checker.py

```python
if __name__ == '__main__':
    _main()
```

Exercises

Exercise 2. (*Word Frequencies*) Implement the function `count_word_frequencies()` in `word_frequencies.py` that takes a list of words as argument and returns a dictionary whose keys are the words from the list and values are the corresponding frequencies. Also implement the function `write_word_frequencies()` that takes a dictionary as argument and writes (in reverse order of values) the key-value pairs of the dictionary to standard output, one per line, and with a ' -> ' between a key and the corresponding value.

```
$ python3 word_frequencies.py
it was the best of times it was the worst of times
<ctrl-d>
was -> 2
it -> 2
times -> 2
the -> 2
of -> 2
worst -> 1
best -> 1
```

Hint: use `dict` method `setdefault()` in the first part and `word_frequencies.keys()` in the second part.

Exercises

word_frequencies.py

```
import operator
import stdio
import sys


# Return a list containing the keys of the dictionary st in
# reverse order of the values of the dictionary.
def keys(st):
    a = sorted(st.items(), key=operator.itemgetter(1),
               reverse=True)
    return [v[0] for v in a]


# Return a dictionary whose keys are the words from the given
# list of words and values are the corresponding frequencies.
def count_word_frequencies(words):
    # Initialize st to an empty dictionary.
    ...

    # Iterate over each word w in words.
    for ... in ...:
        # Add w with frequency 0 to st using the
        # st.setdefault() method.
        ...

        # Increment the frequency of w by 1.
        ...

    # Return st.
    ...


# Write (in reverse order of values) the key-value pairs of
# the dictionary st to standard output, one per line, and with
# a ’ -> ’ between a key and the corresponding value.
def write_word_frequencies(st):
    # Initialize words to the keys in st in reverse order of
    # the values (frequencies).
```

Exercises

word_frequencies.py

```
    ...

    # Iterate over each word w in words.
    for ... in ...:
        # Write w and its frequency with a ’ -> ’ between
        # the two.
        ...


# Test client [DO NOT EDIT].
def _main():
    words = stdio.readAllStrings()
    write_word_frequencies(count_word_frequencies(words))


if __name__ == ’__main__’:
    _main()
```

Exercises

Exercise 3. (*2D Point*) Define a data type `Point` in `point.py` that represents a point in 2D. The data type must support the following API:

method	description
`Point(x, y)`	a new point p from the given x and y values
`p.distanceTo(q)`	the Euclidean distance between p and q
`str(p)`	the string representation of p as `'(x, y)'`

```
$ python3 point.py 0 1 1 0
p1 = (0.0, 1.0)
p2 = (1.0, 0.0)
d(p1, p2) = 1.41421356237
```

Exercises

point.py

```python
import stdio
import sys


class Point:
    """
    Represents a point in 2-dimensional space.
    """

    def __init__(self, x, y):
        """
        Construct a new point given its x and y coordinates.
        """

        ...

    def distanceTo(self, other):
        """
        Return the Euclidean distance between self and other.
        """

        ...

    def __str__(self):
        """
        Return a string representation of self.
        """

        ...


# Test client [DO NOT EDIT].
def _main():
    x1, y1, x2, y2 = map(float, sys.argv[1:])
    p1 = Point(x1, y1)
    p2 = Point(x2, y2)
    stdio.writeln('p1 = ' + str(p1))
    stdio.writeln('p2 = ' + str(p2))
```

Exercises

point.py

```
    stdio.writeln('d(p1, p2) = ' + str(p1.distanceTo(p2)))


if __name__ == '__main__':
    _main()
```

Exercises

Exercise 4. (*1D Interval*) Define a data type `Interval` in `interval.py` that represents a closed 1D interval. The data type must support the following API:

method	description
`Interval(lbound, rbound)`	a new interval i from the given lower and upper bounds for the interval
`i.lbound()`	lower bound of i
`i.ubound()`	upper bound of i
`i.contains(x)`	does i contain the point x?
`i.intersects(j)`	does i intersect the interval j?
`str(i)`	the string representation of i as `'[lbound, rbound]'`

```
$ python3 interval.py 3.14
0 1 0.5 1.5 1 2 1.5 2.5 2.5 3.5 3 4
<ctrl-d>
[2.5, 3.5] contains 3.140000
[3.0, 4.0] contains 3.140000
[0.0, 1.0] intersects [0.5, 1.5]
[0.0, 1.0] intersects [1.0, 2.0]
[0.5, 1.5] intersects [1.0, 2.0]
[0.5, 1.5] intersects [1.5, 2.5]
[1.0, 2.0] intersects [1.5, 2.5]
[1.5, 2.5] intersects [2.5, 3.5]
[2.5, 3.5] intersects [3.0, 4.0]
```

Exercises

interval.py

```python
import stdio
import sys


class Interval:
    """
    Represents a 1-dimensional interval [lbound, rbound].
    """

    def __init__(self, lbound, rbound):
        """
        Construct a new interval given its lower and
        upper bounds.
        """

        ...

    def lbound(self):
        """
        Return the lower bound of the interval.
        """

        ...

    def rbound(self):
        """
        Return the upper bound of the interval.
        """

        ...

    def contains(self, x):
        """
        Return True if self contains the point x and
        False otherwise.
        """

        ...
```

Exercises

```
interval.py
```

```
    def intersects(self, other):
        """
        Return True if self intersects other and False othewise.
        """

        ...

    def __str__(self):
        """
        Return a string representation of self.
        """

        ...

# Test client [DO NOT EDIT].
def _main():
    x = float(sys.argv[1])
    intervals = []
    while not stdio.isEmpty():
        lbound = stdio.readFloat()
        rbound = stdio.readFloat()
        intervals += [Interval(lbound, rbound)]
    for i in range(len(intervals)):
        if intervals[i].contains(x):
            stdio.writef(’%s contains %f\n’, intervals[i], x)
    for i in range(len(intervals)):
        for j in range(i + 1, len(intervals)):
            if intervals[i].intersects(intervals[j]):
                stdio.writef(’%s intersects %s\n’,
                             intervals[i], intervals[j])


if __name__ == ’__main__’:
    _main()
```

Exercises

Exercise 5. (*Rectangle*) Define a data type `Rectangle` in `rectangle.py` that represents a rectangle using 1D intervals (ie, `Interval` objects) to represent its x (width) and y (height) segments. The data type must support the following API:

method	description
`Rectangle(xint, yint)`	a new rectangle r from the given x and y segments (as interval objects)
`r.area()`	the area of r
`r.perimeter()`	the perimeter of r
`r.contains(x, y)`	does r contain the point (x, y)?
`r.intersects(s)`	does r intersect the rectangle s?
`str(r)`	the string representation of r as `'[x1, x2] x [y1, y2]'`

```
$ python3 rectangle.py 1.01 1.34
0 1 0 1 0.7 1.2 .9 1.5
<ctrl-d>
0 1 0 1 0.7 1.2 .9 1.5
Area([0.0, 1.0] x [0.0, 1.0]) = 1.000000
Perimeter([0.0, 1.0] x [0.0, 1.0]) = 4.000000
Area([0.7, 1.2] x [0.9, 1.5]) = 0.300000
Perimeter([0.7, 1.2] x [0.9, 1.5]) = 2.200000
[0.7, 1.2] x [0.9, 1.5] contains (1.010000, 1.340000)
[0.0, 1.0] x [0.0, 1.0] intersects [0.7, 1.2] x [0.9, 1.5]
```

Exercises

rectangle.py

```python
import stdio
import sys
from interval import Interval


class Rectangle:
    """
    Represents a rectangle as two (x and y) intervals.
    """

    def __init__(self, xint, yint):
        """
        Construct a new rectangle given the x and y intervals.
        """

        ...

    def area(self):
        """
        Return the area of self.
        """

        ...

    def perimeter(self):
        """
        Return the perimeter of self.
        """

        ...

    def contains(self, x, y):
        """
        Return True if self contains the point (x, y) and
        False otherwise.
        """

        ...
```

Exercises

rectangle.py

```python
    def intersects(self, other):
        """
        Return True if self intersects other and
        False othewise.
        """

        ...

    def __str__(self):
        """
        Return a string representation of self.
        """

        ...


# Test client [DO NOT EDIT].
def _main():
    x = float(sys.argv[1])
    y = float(sys.argv[2])
    rectangles = []
    while not stdio.isEmpty():
        lbound1 = stdio.readFloat()
        rbound1 = stdio.readFloat()
        lbound2 = stdio.readFloat()
        rbound2 = stdio.readFloat()
        rectangles += [Rectangle(Interval(lbound1, rbound1),
                                 Interval(lbound2, rbound2))]
    for i in range(len(rectangles)):
        stdio.writef('Area(%s) = %f\n', rectangles[i],
                     rectangles[i].area())
        stdio.writef('Perimeter(%s) = %f\n', rectangles[i],
                     rectangles[i].perimeter())
        if rectangles[i].contains(x, y):
            stdio.writef('%s contains (%f, %f)\n',
                         rectangles[i], x, y)
```

Exercises

rectangle.py

```
    for i in range(len(rectangles)):
        for j in range(i + 1, len(rectangles)):
            if rectangles[i].intersects(rectangles[j]):
                stdio.writef('%s intersects %s\n',
                             rectangles[i], rectangles[j])


if __name__ == '__main__':
    _main()
```

Problems

Student

The guidelines for the project problems that follow will be of help only if you have read the description ↗ of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Instructor

Please summarize the project description ↗ for the students before you walk them through the rest of this checklist document.

Problems

Problem 1. (*Markov Model Data Type*) Create a data type `MarkovModel` to represent a Markov model of order k from a given text string, and supporting the following API:

method	**description**
`MarkovModel(text, k)`	create a Markov model `model` of order k from *text*
`model.order()`	order k of Markov model
`model.kgram_freq(kgram)`	number of occurrences of *kgram* in text
`model.char_freq(kgram, c)`	number of times that character c follows *kgram*
`model.rand(kgram)`	a random character following the given *kgram*
`model.gen(kgram, T)`	a string of length T characters generated by simulating a trajectory through the corresponding Markov chain, the first k characters of which is *kgram*

Hints

- ⇝ Instance variables
  - ⇝ Order of the Markov model, `_k` (`int`)
  - ⇝ A dictionary to keep track of character frequencies, `_st` (`dict`)

Problems

- `MarkovModel(text, k)`
  - Initialize instance variables appropriately
  - Construct circular text `circ_text` from `text` by appending the first `k` characters to the end; for example, if `text = 'gagggagaggcgagaaa'` and `k = 2`, then `circ_text = 'gagggagaggcgagaaaga'`
  - For each `kgram` from `circ_text`, and the character `next_char` that immediately follows `kgram`, increment the frequency of `next_char` in the dictionary `_st[kgram]` by 1; for the above example, the dictionary `_st`, at the end of this step, should look like the following:

```
{'aa': {'a': 1, 'g': 1},
 'ag': {'a': 3, 'g': 2},
 'cg': {'a': 1},
 'ga': {'a': 1, 'g': 4},
 'gc': {'g': 1},
 'gg': {'a': 1, 'c': 1, 'g': 1}}
```

- Exercise: suppose `text = 'shesellsseashellsontheseashore'` and `k = 2`.
  - What is the value of `circ_text`?
  - What does the dictionary `_st` contain?

Problems

- `model.order()`
  - Return the order of the Markov model
- `model.kgram_freq(kgram)`
  - Return the frequency of `kgram`, which is simply the sum of the values of `_st[kgram]`
- `model.char_freq(kgram, c)`
  - Return the number of times `c` immediately follows `kgram`, which is simply the value of `c` in `_st[kgram]`
- `model.rand(kgram)`
  - Use `stdrandom.discrete()` to randomly select and return a character that immediately follows `kgram`
- `model.gen(kgram, T)`
  - Initialize a variable `text` to `kgram`
  - Perform `T - _k` iterations, where each iteration involves appending to `text` a random character obtained using a call to `self.rand()` and updating `kgram` to the last `_k` characters of `text`
  - Return `text`

Problems

Problem 2. (*Random Text Generator*) Write a client program `text_generator.py` that takes two command-line integers k and T, reads the input text from standard input and builds a Markov model of order k from the input text; then, starting with the k-gram consisting of the first k characters of the input text, prints out T characters generated by simulating a trajectory through the corresponding Markov chain, followed by a new line.

Hints

- Read command-line arguments `k` and `T`
- Initialize `text` to text read from standard input using `sys.stdin.read()`
- Create a Markov model `model` using `text` and `k`
- Use `model.gen()` to generate a random text of length `T` and starting with the first `k` characters of `text`
- Write the random text to standard output

Problems

Problem 3. (*Noisy Message Decoder*) Write a client program `fix_corrupted.py` that takes an integer k (model order) and a string s (noisy message) as command-line arguments, reads the input text from standard input, and prints out the most likely original string.

Hints

- Implement `fix_corrupted.py` as follows:
  - Read command-line arguments `k` and `s`
  - Initialize `text` to text read from standard input using `sys.stdin.read()`
  - Create a Markov model `model` using `text` and `k`
  - Use `model.replace_unknown()` to decode the corrupted text `s`
  - Write the decoded text to standard output
- Implement the method `replace_unknown()` in `MarkovModel` using the idea suggested on the following slide

Problems

⇝ When we fix the corrupted messages, we have to look at the missing letter in the context of what comes before it and what comes after it

Example: suppose the corrupted text is `'it w~s th'`, `k = 4`, and the characters that follow the 4-gram `'it w'` (ie, potential replacement characters for `'~'`) are `'a'`, `'b'`, and `'c'`

We refer to the replacement characters as hypotheses H_a, H_b, and H_c, and the goal is to pick the best hypothesis to replace `'~'`

We use the notation `'abcd'|'e'` to mean the probability of finding an `'e'` after the 4-gram `'abcd'`; this probability is 0 if `'e'` does not follow `'abcd'` in the text

The likelihood of H_a is the product of 5 probabilities: `'it w'|'a'`, `'t wa'|'s'`, `' was'|' '`, `'was '|'t'`, and `'as t'|'h'`

The likelihood of H_b is the product of 5 probabilities: `'it w'|'b'`, `'t wb'|'s'`, `' wbs'|' '`, `'wbs '|'t'`, and `'bs t'|'h'`

The likelihood of H_c is the product of 5 probabilities: `'it w'|'c'`, `'t wc'|'s'`, `' wcs'|' '`, `'wcs '|'t'`, and `'cs t'|'h'`

Now, the character that will replace `'~'` with is the one with the maximum likelihood; for example, if $\max(H_a, H_b, H_c) = H_a$, then `'~'` is replaced by `'a'`; use the `argmax()` function for this purpose

Problems

- Exercise: suppse `text = ’shesellsseashellsontheseashore’`, `k = 2`, and `corrupted = ’sh~ore’`.
  - What are the replacement characters (ie, hypotheses) for ’~’?
  - What is the likelihood of each hypothesis?
  - What is the best hypothesis (ie, the character that will replace ’~’)?
- Pseudocode for `model.replace_unknown()`

```
if corrupted[i] == ’~’:
    kgram_before = kgram before ~
    kgram_after = kgram after ~
    probs = []
    for each hypothesis from hypotheses (replacements for ~):
        context = kgram_before + hypothesis + kgram_after
        p = 1.0
        for i from 0 to _k + 1:
            kgram = kgram from context starting at i
            char = character from context that follows kgram
            if kgram or char is non-existent, then set p to 0
            and break; otherwise, multiply p by probability of
            char following kgram
        append p to probs
    append to original the hypothesis that maximizes probs
```

Problems

Be sure to test your programs thoroughly using the input files under the `data` directory

```
$ ls data
aesop.txt                 biden.txt             monalisa.txt
amendments.txt            deadend.txt           obama.txt
barack-obama2004dnc.txt   input17.txt           palin.txt
bbbabbabbbbaba.txt        input53.txt           pearl_jam.txt
Beatles.txt               IolantheLibretto.txt  wiki_100k.txt
bible.txt                 mccain.txt            zell-millier2004rnc.txt
```

For example

```
$ python3 text_generator.py 5 50 < data/Beatles.txt
Words you, I don't be a catch Will you be very con
```

```
$ python3 fix_corrupted.py 3 "she s~lls sea s~ells on th~ sea s~ore" < data/wiki_100k.txt
she sells sea spells on the sea store
```

Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include

- Time (in hours) spent on the project
- Difficulty level (1: very easy; 5: very difficult) of the project
- A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- Acknowledgement of any help you received
- Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files

- Make sure your programs meet the style requirements by running the following command on the terminal

  ```
  $ pycodestyle <program>
  ```

  where `<program>` is the `.py` file whose style you want to check
- Make sure your programs meet the input and output specifications by running the following command on the terminal

  ```
  $ python3 run_tests.py -v [<items>]
  ```

  where the optional argument `<items>` lists the exercises/problems (`Exercise1`, `Problem2`, etc.) you want to test, separated by spaces; all the exercises/problems are tested if no argument is given
- Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit

1. `password_checker.py`
2. `word_frequencies.py`
3. `point.py`
4. `interval.py`
5. `rectangle.py`
6. `markov_model.py`
7. `text_generator.py`
8. `fix_corrupted.py`
9. `report.txt`