LAST LAB OF THE SEMESTER!!!!!!!!!!!!

(It's also extra credit!)
TODAY’S LESSON

n-grams

(Generating reasonable text by training a model on historical text)
Oracle Demo

Using multiple instances of a text-generating class to generate text for different characters in the same book.
Hagrid: "member . i should n'ta told yeh that ! he blurted out . i want . yer ticket fer hogwarts he said ."

Harry: " n't talk . i tried to turn him yellow yesterday to make him id - he 's runnin ' back up ter the school .”

Vernon: "aid uncle vernon " so all aboard ! " where 's the cannon ? you are boy . platform nine - platform ten ."

Quirrell: "ee the stone...i 'm presenting it to my master...but where is vering treble either but cold and sharp ."

Dumbledore: "dungeons between you and professor quirrell is a complete secret prises even me sometimes...now enough questions ."
The distribution, from our data, of letters that can follow a given letter sequence.

We can use this to randomly generate similar text as the original.
1. Randomly select a n-1 gram (on the left)
2. Let’s say we select ‘i’
3. Given i, the next letter can be
   1. either ‘l’ or ‘a’ with 50% chance each
4. Let’s say we randomly pick ‘a’
5. Now, given ‘a’, what can we choose?
   1. Only m!
6. And then s follows
    1. Only m!
7. s doesn’t have an entry, so what do we do?
   ▪ We can randomly pick a new letter, now we have a new n-1 gram!
   ▪ ...start again from the top, until we decide to stop!

Given more data, our output (blue) will have more possible outcomes.
• We’ve been working with 2-grams or bigrams

• There’s also trigrams:
  ▪ wil, ill, lli, lia, iam, ams

• Which we can turn into a distribution as follows
  ▪ wi → l, il → l, ll → i, li → a, ia → m, am → s

• 4-grams, 20-grams, etc. etc.

• We call these “n-grams”
What will we need to build our text-generating ORACLE ??

- Choose an n for our n-gram
- Some text to build the letter distribution ➔ file input!
- A data structure to hold the letter distribution/fingerprint
- Somewhere to start generating new text
- Something to do when we run out of letters (i.e. what comes after the ‘s’ in Williams?)
  - We’ll need to store our text
Why use classes instead of a pile of functions?

• Encapsulation! Abstraction!

• Maintaining state
  • But we must write our methods to maintain that state
Why use classes instead of a pile of functions?

• Multiple oracles at the same time!

```python
ha = Oracle()
...
print(next(ha.lines()))

dr = Oracle()
...
print(next(dr.lines()))

he = Oracle()
...
print(next(he.lines()))
```
QUESTIONS?

Please contact me!
Oracle Lab: Starter Code

Introduction to Computer Science
Iris Howley
(Generating reasonable text by training a model on historical text)
ENCAPSULATING DATA IN CLASSES TO GENERATE HUMAN-LIKE TEXT

Classes + Dictionaries + Generators + Files
Look at oracle.py

• __slots__ = ['_corpus', '_dist', '_n']
  ▪ A special list to hold the class’ attributes
  ▪ It restricts the attributes to just these!

• __XXXX__ are special python variables/functions
  ▪ __name__, __all__, __slots__, many others!
  ▪ _XXX are variables/functions we don’t want to be public
  ▪ Won’t show up in pydoc3, etc. Just for our use in this Oracle class!
How we interact with Oracle

```python
# When we create an instance of a class, that class's __init__() method is called

g = Oracle()
lines = [line.strip() for line in open('tom.txt')]
text = ' '.join(lines)
g.scan(text)
for line in g.lines():
    print(line)
```
```python
def __init__(self):

    # self is always passed to class methods

    # self refers to this particular object (i.e., an object reference)

    # When we see self.something, we know it’s a variable, method, etc. associated with a particular instance of a class
```
```python
def __init__(self):

def __init__(self, n = 4):
    """Initialize the oracle with n-gram size n."""
    self._n = n
    self._dist = dict()
    self._corpus = ""
```

```
__init__ sets the initial values for attributes within the instance of the class
```
```python
def __init__(self):

def __init__(self, n = 4):
    
    """Initialize the oracle with n-gram size n.""
    self._n = n
    self._dist = dict()
    self._corpus = ""

Can be used to set default values in case the user doesn’t pass an argument
```
Oracle Lab: Using the Oracle

Introduction to Computer Science
Iris Howley
How we interact with Oracle

```python
g = Oracle()
g.scan(text)
for line in g.lines():
    print(line)
```

What does this line imply about Oracle’s `lines()` method?
for line in myoracle.lines():

• `.lines()` is a generator
  – yields lines instead of returning lines!

• Our `.lines()` function will produce a generated line of text that will fit on a single line on the console (70-80 characters)

• But we still need a way to generate individual letters to put in the line
  – for `<variable>` in `<sequence>`
  – `__iter__(self)` function is called on the `<sequence>` object
    o Review Lecture 13/14 on Iterators (and review generators, too)
  – It also yields an element from the sequence, one at a time
How we interact with Oracle

```python
g = Oracle()
g.scan(text)
for line in g.lines():
    print(line)
```

When does this stop printing lines?
WHEN WE HAVE AN INFINITE GENERATOR, HOW TO PRINT LIMITED NUMBER OF VALUES?
Printing Limited Values from Infinite Generator

• Will count off even numbers forever:
  >>> def countEveryOther():
  ...    current = 0
  ...    while True:
  ...      yield current
  ...    current +=2

• Print the values from the generator:
  >>> g = countEveryOther()
  >>> for num in g:
  ...    print(num)

• Will print even numbers infinitely (too fast to read)!!
Printing Limited Values from Infinite Generator

• Will count off even numbers forever:
  >>> def countEveryOther():
  ...    current = 0
  ...    while True:
  ...      yield current
  ...    current +=2

• Print the first 10 values from the generator:
  >>> g = countEveryOther()
  >>> for _, num in zip(range(10), g):
  ...    print(num)

• Built-in zip(…) function zips iterable objects together!
zip(iterable1, iterable2)

>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]

Note that `zip(...)` produces a sequence of tuples:

(iterable1[0], iterable2[0])…
zip(iterable1, iterable2)

>>> x = (1, 2, 3, 4, 5, 6)
>>> y = {7, 8, 9}
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 8), (2, 9), (3, 7)]

Note that zip(..) only produces these paired tuples until one of the iterables parameters runs out of items!
How we interact with Oracle

```python
import oracle

g = Oracle()
g.scan(text)
for _, line in zip(range(1000), g.lines()):
    print(line)

This prints only the first 1000 lines of our Oracle-generated text!
```
QUESTIONS?

Please contact me!
Leftover Slides
Classes

```python
>>> from oracle import Oracle
>>> o = Oracle()
• >>> type(o)
  • <class 'oracle.Oracle'>

• o is an instance of the class, Oracle
• Classes are user-defined types
>>> from oracle import Oracle
>>> o = Oracle()
>>> o
<oracle.Oracle object at 0x103485e48>

...Define the __repr__() function in the oracle class

>>> from oracle import Oracle
>>> o = Oracle()
>>> o
REPR(): Oracle(n=4)
Selecting an item from a sequence

```python
>>> from random import choice
>>> l = ['a', 'b', 'c', 'd']
>>> print(choice(l))
b
>>> print(choice(l))
d
>>> print(choice(l))
a
>>> print(choice(l))
c
>>> print(choice(l))
b
>>> s = "The mountains!"
>>> print(choice(s))
T
>>> print(choice(s))
a
>>> print(choice(s))
!
```
**Shannon Entropy**

\[ H = - \sum_{i=0}^{N-1} p_i \log_2 p_i \]

- Average rate at which information is produced by our data
  - The unexpectedness of a sequence of characters we select

- The entropy of a random variable is calculated with this formula:
  1. Where \( p_i \) is the probability of seeing a given n-gram in our data
  2. Given a set of \( n \) observations
    - Where each observation is a different sequence of characters observed in our data
  3. Compute \( p_i \) for the range all observations multiply by \( \log_2(p_i) \)
  4. Sum across all values
```
__slots__ = []

>>> class Yesteryears:
...     """demo of classes from last week """
... 
... >>> yy = Yesteryears()
... >>> yy.start = 2018
... >>> yy.end = 2022
... >>> yy.mid = 2020
... >>> yy.whatev = "I do what I want!"
... """

>>> class Years:
...     __slots__ = ['start','end']
... 
... >>> newy = Years()
... >>> newy.start = 2017
... >>> newy.end = 2021
... >>> newy.mid = 2019
... Traceback (most recent call last):
...     File "<stdin>", line 1, in <module>
...     AttributeError: 'Years' object has no attribute 'mid'
```
Classes

```python
>>> Class Years:
    """ Define some attributes """
>>> y = Years()
>>> y
   __main__.Years object at 0x108c63860>
>>> from oracle import Oracle
This is __name__!!!!
>>> o = Oracle()
>>> o
   oracle.Oracle object at 0x103485e48>
```