Connect Four (Wrap up)
Final Project

Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores - one for each column!!
    """
    scores = [50] * board.width
    for col in range(board.width):

        ???




    return scores
```

Suppose you're playing with LA 2...

For each column:

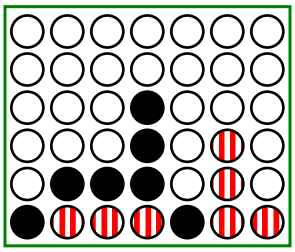
1) add a checker to it

2) ask an opponent with LA 1 for its scores for the resulting board!

3) assume the opponent will makes its best move, and determine your score accordingly

4) remove checker!

scores_for



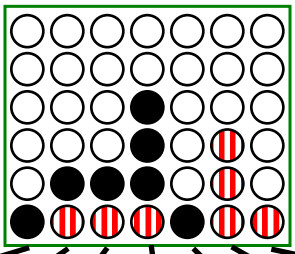
(self) 'X'

possible next move

Suppose you're playing with LA 2...

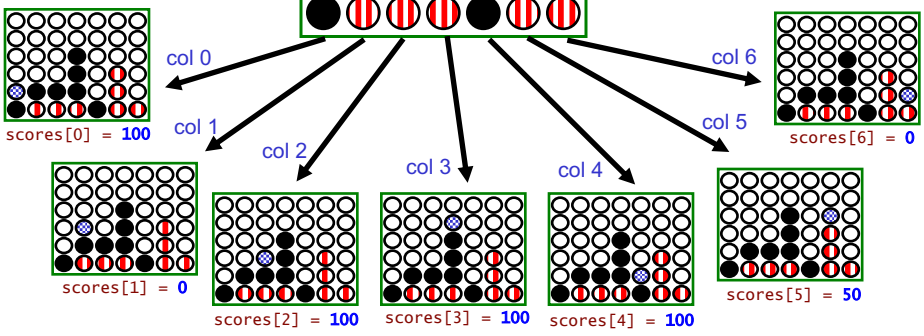
We've tried all columns!

scores_for



(self) 'X'

possible next move




```
return [100, 0, 100, 100, 100, 50, 0]
```

scores_for – the AI in AIPlayer!

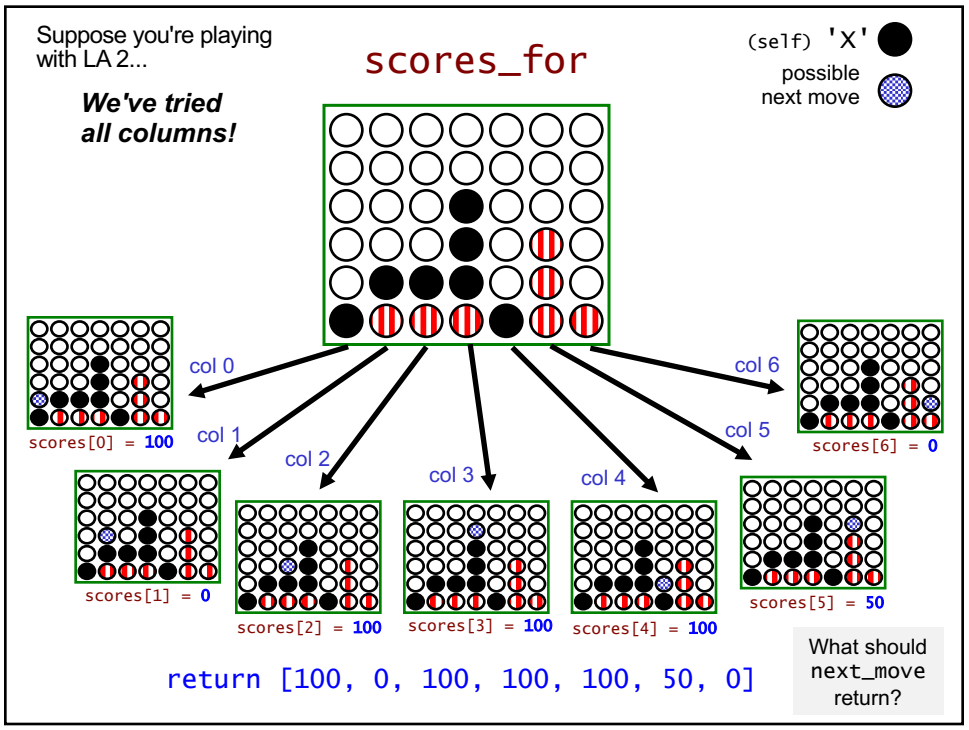
```
def scores_for(self, board):
    """ MUST return a list of scores - one for each column!!
    """
    scores = [50] * board.width
    for col in range(board.width):

        ???




    return scores
```

scores_for – the AI in AIPlayer!

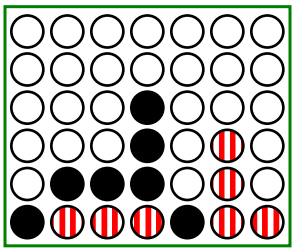
```
def scores_for(self, board):
    """ MUST return a list of scores - one for each column!!
    """
    scores = [50] * board.width
    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col - adding a checker to it
            create an opponent with self.lookahead - 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker

    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores - one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col - adding a checker to it
            create an opponent with self.lookahead - 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```



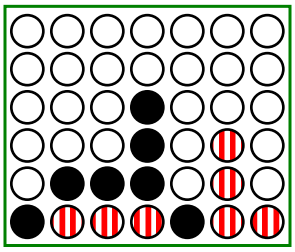
Breaking Ties



```
0   1   2    3   4   5   6
return [100, 0, 100, 100, 100, 50, 0]
```

- possible moves: `???`

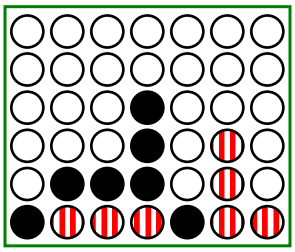
Breaking Ties



```
0   1   2    3   4   5   6
return [100, 0, 100, 100, 100, 50, 0]
```

- possible moves: `[0, 2, 3, 4]`

Breaking Ties



0 1 2 3 4 5 6

```
return [100, 0, 100, 100, 100, 50, 0]
```

- possible moves: `[0, 2, 3, 4]`
- `self.tiebreak == 'LEFT': return 0`
- `self.tiebreak == 'RIGHT': return 4`
- `self.tiebreak == 'RANDOM': choose at random!`

What's Left in CS 111

S	M	T	W	T	F	S
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21

Final Project

- posted today!
- parts I and II due 12/5 (part of PS 10)
- full project due 12/10

Problem Set 9

- late part I today
- part II due 11/24

Problem Set 10

- parts I and II of final project
- bit of CS theory
- all due **12/5**

Final exam:

- 12/19, 9-11 am

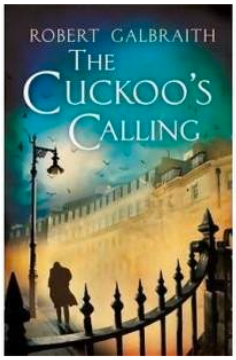
Final Project

- Worth 150 points
- ***Cannot*** be replaced by the final exam
- More room for creativity than a usual assignment
- Pair-optional
  - pairs *must* work together, *in the same place*
  - they must share the work equally
  - see the collaboration policy

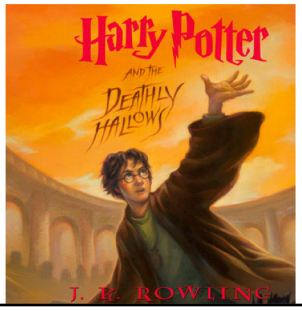
Text Modeling and Classification

Though Robin Ellacott's twenty-five years of life had seen their moments of drama and incident, she had never before woken up in the certain knowledge that she would remember the coming day for as long as she lived.

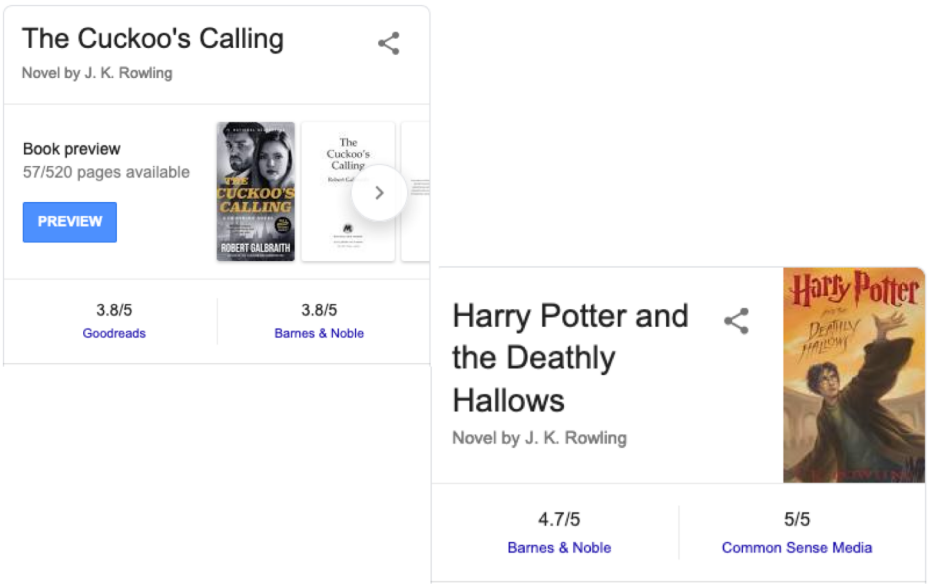
– first paragraph of *The Cuckoo's Calling* by Robert Galbraith



a 2013 crime fiction novel by Robert Galbraith



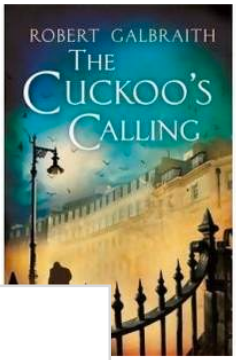
Text Modeling and Classification



Text Modeling and Classification

Though Robin Ellacott's twenty-five years of life had seen their moments of drama and incident, she had never before woken up in the certain knowledge that she would remember the coming day for as long as she lived.

– first paragraph of *The Cuckoo's Calling*



by

FRIDAY, AUG 23, 2013 08:20 AM EDT

How J.K. Rowling was exposed as Robert Galbraith

A mathematical analysis of "The Cuckoo's Calling" revealed the "Harry Potter" author's linguistic signature

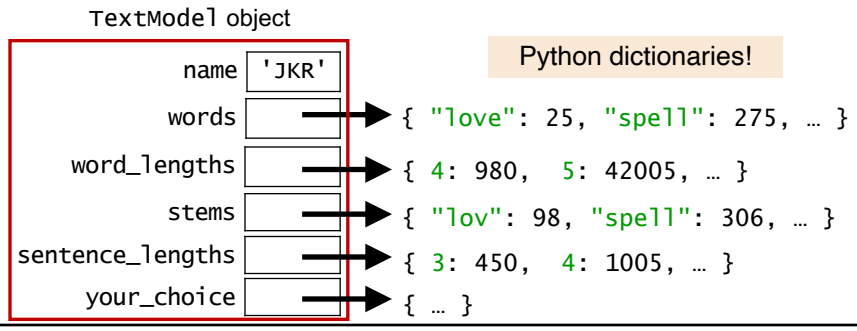
PATRICK JUOLA, SCIENTIFIC AMERICAN

Text Modeling and Classification

- Build a model of a *body of text*.
  - works by an author / of a certain genre / etc.
  - articles from a given publication / type of publication
  - scripts from a given TV series
  - etc.
- Implement a *similarity score* that allows you to compare two bodies of text.
  - room for creativity here!
- ***You pick some bodies of text and perform comparisons!***

Modeling a Body of Text

- Based on features of the text.
  - word-frequencies
  - word-length frequencies
  - stem-frequencies
  - sentence-length frequencies
  - *one other feature of your choice!*



Final Project

- You already know enough to complete Parts I and II.
  - OOP
  - dictionaries
  - file-processing
- We'll discuss Parts III-V soon.
  - the project write-up also includes more detail

Maurice Wilkes

- Pioneering computer scientist
- Built the first working example of a computer that stored its programs in memory.
  - following von Neumann's ideas
- Made other important advances
- Won the Turing Award (computer science's Nobel Prize)
- Has this famous quote:

  I well remember when this realization first came on me with full force … that a good part of the remainder of my life was going to be spent ***in finding errors in my own programs.***

Getting Good at Debugging

- Trace through your code on paper – make a table!
- Add temporary `print` statements to see how variables change.
- Watch out for subtle bugs!
  - logic errors – your algorithm isn't correct
  - syntax errors (ones involving the rules/format of the language) that Python doesn't catch

Getting Good at Debugging (cont.)

- Example: a client function `random_move(b, checker)`
  - adds `checker` to a randomly selected non-full column in b
  - returns the index of the selected column, or -1 if b is full

```
>>> b = Board(2, 2)
>>> random_move(b, 'O')
1
>>> b
| | |
| |O|
-----
 0 1
>>> b.add_checkers('001')   # fill up the rest of the board
>>> b
|O|X|
|X|O|
-----
 0 1
>>> random_move(b, 'X')
-1
```

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col == random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. two lines
D. more than three lines

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col == random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. **two lines**
D. more than three lines

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col == random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. **two lines**
D. more than three lines

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full() == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col == random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. **two lines**
D. more than three lines

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full() == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col == random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. **two lines**
D. more than three lines

How many lines have bugs in them? (which ones?)

```
def random_move(b, checker):
    """ adds the specified checker to a randomly
        selected column col in board b; returns col
    """
    if b.is_full() == True:
        return -1    # no column has room

    # keep choosing until you get a non-full column
    possible_cols = range(b.width)
    col = random.choice(possible_cols)
    while b.can_add_to(col) == False:
        col = random.choice(possible_cols)

    b.add_checker(checker, col)
    return col
```

A. one line
C. three lines
B. **two lines**
D. more than three lines

Recall: Inheritance in PS 9

- `Player` – a class for human Connect Four players
  - includes fields and methods needed by all C4 players
  - in particular, a `next_move` method
- `RandomPlayer` – a class for an *un*intelligent computer player
  - no new fields
  - **overrides `next_move`** with a version that chooses at random from the non-full columns
- `AIPlayer` – a class for an "intelligent" computer player
  - uses AI techniques
  - new fields for details of its strategy
  - **overrides `next_move`** with a version that tries to determine the best move!

Using the Player Classes

- Example 1: two human players

```
>>> connect_four(Player('X'), Player('O'))
```

- Example 2: human player vs. AI computer player:

```
>>> connect_four(Player('X'), AIPlayer('O', 'LEFT', 3))
```

- `connect_four()` repeatedly calls `process_move()`:

```
def connect_four(p1, p2):
    print('Welcome to Connect Four!')
    print()
    b = Board(6, 7)
    print(b)

    while True:
        if process_move(p1, b) == True:
            return b
        if process_move(p2, b) == True:
            return b
```

OOP == Object-Oriented Power!

```
def process_move(p, b):
    ...
    col = p.next_move(b)

    ...
```

- Which version of `next_move` gets called?

OOP == Object-Oriented Power!

```
def process_move(p, b):
    ...
    col = p.next_move(b)

    ...
```

- Which version of `next_move` gets called?
- It depends!
  - if `p` is a `Player` object, call `next_move` from that class
  - if `p` is a `RandomPlayer`, call that version of `next_move`
  - if `p` is an `AIPlayer`, call that version of `next_move`
- The appropriate version is automatically called!

Beware!

- Correct approach: call the `next_move` method within the object to which the variable `p` refers:

```
def process_move(p, b):
    ...
    col = p.next_move(b)
    ...
```

- In theory, we can treat `next_move` as if it were a function:

```
def process_move(p, b):
    ...
    col = Player.next_move(p, b)       # wrong!
    ...
```

- This won't work! Why?
  It *always* calls the `Player` version of `next_move`.
  It *never* calls the `RandomPlayer` or `AIPlayer` version!

What's Left in CS 111

S	M	T	W	T	F	S
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21

Final Project

- posted today!
- parts I and II due 12/5 (part of PS 10)
- full project due 12/10

Problem Set 9

- late part I today
- part II due 11/24

Problem Set 10

- parts I and II of final project
- bit of CS theory
- all due **12/5**

Final exam:

- 12/19, 9-11 am