lesson3_naivebayes_exercises

April 29, 2018

```
In [1]: # Run this every time you open the spreadsheet
        %load_ext autoreload
        %autoreload 2

        from collections import Counter
        import lib
```

1 Load the content of the boxes

```
In [2]: # Load the data.
        # This function returns box_a and box_b, both lists of colors of the balls in each box
        box_a, box_b = lib.get_box_contents()

        print("Number of balls in box A: %d" % len(box_a))
        print("Number of balls in box B: %d" % len(box_b))
```

```
Number of balls in box A: 100
Number of balls in box B: 100
```

2 Python concepts

Let's review and look at some new Python concepts before we implement the box and ball examples.

2.0.1 Dictionaries

In Python, a *dict* is a collection of items in which each element can be accessed by a *key*. The *key* is typically a string and the items can be of any data type, e.g., booleans, integers, strings. Each key can be used for only one item.

You can create dictionaries like this:

```
west_coast_state_capitals = {"California": "Sacramento", "Oregon": "Salem", "Washington": "Olym
prices = {"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75, "avocado": 0.85, "bread": 3.
```

To access a value in a dictionary, use the name of the dictionary and put they *key* in squared brackets:

```
west_coast_state_capitals["California"] # returns "Sacramento"
prices["milk"] # returns 2.00
```

```
In [3]: # Exercise 1.
        # Create a dictionary called "authors" that maps the following book titles to their
        authors.
        #
        # Harry Potter - J.K. Rowling
        # The Casual Vacancy - J.K. Rowling
        # The Hunger Games - Suzanne Collins
        # Never Let Me Go - Kazuo Ishiguro
        # The Catcher in the Rye - J.D. Salinger
        #
        # Then print the author of "The Catcher in the Rye" and "Harry Potter" using your
        dictionary.

        #### YOUR CODE STARTS HERE ####
        authors={"Harry Potter":"J.K. Rowling","The Casual Vacancy":"J.K. Rowling","The Hunger
        Games":"Suzanne Collins"
                 ,"Never Let Me Go":"Kazuo Ishiguro","The Catcher in the Rye":"J.D. Salinger"}

        #### YOUR CODE ENDS HERE ####

        print("CORRECT" if authors["Harry Potter"] == "J.K. Rowling" else "INCORRECT")
        print("CORRECT" if authors["The Casual Vacancy"] == "J.K. Rowling" else "INCORRECT")
        print("CORRECT" if authors["Never Let Me Go"] == "Kazuo Ishiguro" else "INCORRECT")

CORRECT
CORRECT
CORRECT
```

Adding, updating, and deleting items from dictionaries You can also add, change and delete items after you created an dictionary.

For example, the following code creates an empty dictionary *prices* and then adds two items to it.

```
prices = {}
prices["milk"] = 2.00
prices["avocado"] = 0.85

print prices # outputs {'avocado': 0.85, 'milk': 2.0}
```

To update an item in a dictionary, simply assign a new value to it:

```
prices = {}
prices["milk"] = 2.00
prices["avocado"] = 0.85
print prices # outputs {'avocado': 0.85, 'milk': 2.0}


prices["milk"] = 2.25
print prices # outputs {'avocado': 0.85, 'milk': 2.25}
```

Tto delete an item from a dictionary, use the *del* keyword as in the following snippet:

```
prices = {}
prices["milk"] = 2.00
prices["avocado"] = 0.85
```

```
print prices # outputs {'avocado': 0.85, 'milk': 2.0}

del prices["milk"]
print prices # outputs {'avocado': 0.85}
```

Iterating through dictionaries You can also iterate through all items in a dictionary with a for-loop. When you loop through a dictionary, the key is assigned to the loop variable during each iteration.

```
prices = {"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75}

for product in prices:
   print product, prices[product]
   # Sidenote: You can print multiple variables in the same line
   # by separating them with a comma.
```

This program outputs something like: (order may vary)

```
spaghetti 2.5
milk 2.0
peanut butter 2.75
```

```
In [4]: # Imagine you are running a store that sells spaghetti, milk, peanut butter, avocados,
        and bread, and you
        # store the prices for these products in the following dictionary.

        prices = {"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75, "avocado": 0.85,
        "bread": 3.25}

        # Exercise 2(a).

        # Your distributor increased prices on all products by 25 cents, so you'll have to
        increase your prices
        # by 25 cents as well. Increase every value in the prices dictionary by 25 cents.

        #### YOUR CODE STARTS HERE ####

        for product in prices:
            prices[product]+=0.25
            pass

        #### YOUR CODE ENDS HERE ####

        print(prices)

        print("CORRECT" if prices["spaghetti"] == 2.75 else "INCORRECT")
        print("CORRECT" if prices["milk"] == 2.25 else "INCORRECT")
        print("CORRECT" if prices["bread"] == 3.50 else "INCORRECT")
```

```
{'spaghetti': 2.75, 'milk': 2.25, 'peanut butter': 3.0, 'avocado': 1.1, 'bread': 3.5}
CORRECT
CORRECT
CORRECT
```

```
In [5]: # Exercise 2(b).

        # You added bananas to your inventory and you sell them for 95 cents. Add a new entry
        # for bananas to the prices dictionary.

        #### YOUR CODE STARTS HERE ####
        prices["bananas"]=0.95

        #### YOUR CODE ENDS HERE ####

        print(prices)
        print("CORRECT" if prices["bananas"] == 0.95 else "INCORRECT")
```

```
{'spaghetti': 2.75, 'milk': 2.25, 'peanut butter': 3.0, 'avocado': 1.1, 'bread': 3.5,
'bananas': 0.95}
CORRECT
```

```
In [6]: # Exercise 2(c).

        # You are no longer selling peanut butter. Remove the entry for peanut butter from
        prices.

        #### YOUR CODE STARTS HERE ####
        del prices["peanut butter"]

        #### YOUR CODE ENDS HERE ####

        print(prices)
```

```
{'spaghetti': 2.75, 'milk': 2.25, 'avocado': 1.1, 'bread': 3.5, 'bananas': 0.95}
```

2.0.2 Counter

Python comes with a special dictionary type, the `Counter` type, which makes it easier to work with counts.

A Counter works just like a dictionary but instead of giving an error when you use a key for which no entry exists, it will return 0.

To use Counters, you first have to run the following import statement.

```
from collections import Counter
```

Let's create a Counter to keep track of how many birds I've seen. Below, I create a new empty Counter called `bird_counter`. Note that if I ask the counter how many sparrows I've seen, it returns 0 even though `"sparrow"` is not in the keys.

```
bird_counter = Counter()
print bird_counter["sparrow"] # outputs 0
```

I just spotted a finch! Let's increment the counter.

```
bird_counter["finch"] += 1
print bird_counter["finch"] # outputs 1
```

```
In [7]: # Exercise 3.
        # Write code that counts how many of each type of fruit there are in the fruit basket.
        # Use a Counter to store the counts and print the final Counter object.
```

```
        # Hint: Use a for-loop to iterate through all the items in fruit_basket.

        from collections import Counter

        fruit_basket = ["apple", "banana", "plum", "apple", "apricot", "plum", "apple", "apple",
        "apricot", "apricot"]

        #### YOUR CODE STARTS HERE ####
        fruit_counter = Counter()
        for fruit in fruit_basket:
            fruit_counter[fruit] += 1

        print(fruit_counter)


        #### YOUR CODE ENDS HERE ####
```

```
Counter({'apple': 4, 'apricot': 3, 'plum': 2, 'banana': 1})
```

2.0.3 Turning lists into Counters

Counters come with several other useful features. One of them is that you can automatically turn a list into a counter. For example, the following snippet counts how many of each letter there are in the list my_letters.

```
my_letters = ["a", "b", "b", "c", "c", "c", "d", "d", "d", "d"]
letter_counter = Counter(my_letters)

print letter_counter # outputs Counter({'d': 4, 'c': 3, 'b': 2, 'a': 1})
```

```
In [8]: # Exercise 4.
        # Re-implement the program from Exercise 3 without using a for-loop.

        from collections import Counter

        fruit_basket = ["apple", "banana", "plum", "apple", "apricot", "plum", "apple", "apple",
        "apricot", "apricot"]

        #### YOUR CODE STARTS HERE ####
        fruit_counter = Counter(fruit_basket)
        print(fruit_counter)

        #### YOUR CODE ENDS HERE ####
```

```
Counter({'apple': 4, 'apricot': 3, 'plum': 2, 'banana': 1})
```

2.0.4 Iterating through counters

You can iterate through a Counter just like a dictionary. If you use a for-loop with a Counter, it will loop through all keys.

```
prices = Counter({"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75})

for product in prices:
  print product
```

This program will print something like: (the order may vary)

```
spaghetti
milk
peanut butter
```

You can also get a list of all values stored in a Counter using the *.values()* method.

```
prices = Counter({"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75})

vals = prices.values()

print vals # outputs [2.5, 2.0, 2.75] (the order may vary)
```

```
In [9]: # Exercise 6.

        # The following counter stores how many rooms of each type a hotel has.

        hotel_rooms = Counter({"1 queen-sized bed": 25, "1 king-sized bed": 14,
                               "2 queen-sized beds": 12, "Honeymoon suite": 1,
                               "Presidential suite": 1})

        # Write some code that prints each room type and how many rooms of each type there are.
        e.g., "Presidential suite 1"


        #### YOUR CODE STARTS HERE ####

        for room, count in hotel_rooms.items():
            print(room, count)

        #### YOUR CODE ENDS HERE ####
```

```
1 queen-sized bed 25
1 king-sized bed 14
2 queen-sized beds 12
Honeymoon suite 1
Presidential suite 1
```

2.0.5 Computing the sum of a list of numbers

Sometimes it can also be really useful to compute the sum of a list of numbers. For example, assume that the following list stores the weights of products in a package and you want to compute the total weight of the package.

```
weights = [3, 4, 5, 1, 2, 9, 12, 11]
```

Python comes with a function `sum` that allows you to quickly sum over a list of numbers.

```
total_weight = sum(weights)
print total_weight # outputs 47
```

You can also use this function together with the values of a Counter. For example, the following code computes how much it would cost if one bought every item in the prices Counter.

```
prices = Counter({"spaghetti": 2.50, "milk": 2.00, "peanut butter": 2.75})

vals = prices.values()

total = sum(vals)

print total # outputs 7.25
```

```
In [10]: # Exercise 6(a).
         # Compute the sum of all numbers from 1 to 10 and assign to the variable "total"

         #### YOUR CODE STARTS HERE ####

         numbers = list(range(1, 11))
         total = sum(numbers)

         #### YOUR CODE ENDS HERE ####

         print(total)
         print("CORRECT" if total == 55 else "INCORRECT")
```

```
55
CORRECT
```

```
In [11]: # Exercise 6(b).
         # Store the counts of each type of pet in a Counter and use that counter to compute the
         total number of pets.
         # Save the result in the variable "total"

         my_pets = ['cat', 'lizard', 'cat', 'dog', 'cat', 'snake', 'dog', 'cat', 'dog', 'parrot']

         #### YOUR CODE STARTS HERE ####

         pet_counter = Counter(my_pets)
         vals = pet_counter.values()
         total = sum(vals)

         #### YOUR CODE ENDS HERE ####

         print(total)
         print("CORRECT" if total == 10 else "INCORRECT")
```

```
10
CORRECT
```

2.0.6 Dividing integers in Python

One of the pecularities of Python (and some other programming languages) is that if you divide two integers, it will always return the results rounded down to the next integer and never a decimal number.

For example, if you compute $1/2$, it will return 0.

This can be particularly problematic when we are dealing with fractions or percentatages, as we often do when we compute probabilities. The easiest way to get around this is by turning one of the two numbers into a decimal number with the function *float*. This will change the representation of the number from an integer to a decimal number and when you then run the division, it will return a decimal. For example, consider the following two divisions:

```
res1 = 1/10

res2 = float(1)/10

print res1
print res2
```

This program will produce the following output:

```
0
0.1
```

```
In [12]: # Exercise 7.
         # Divide each number in the following list by 2 and print it.

         numbers = [3, 5, 6, 7, 9]

         #### YOUR CODE STARTS HERE ####

         for number in numbers:
             print(number / 2)

         #### YOUR CODE ENDS HERE ####

         # The output should be 1.5, 2.5, 3.0, 3.5, 4.5
```

```
1.5
2.5
3.0
3.5
4.5
```

```
In [13]: # Exercise 8.
         # Compute the fraction of each type of animal (e.g., the fraction of lizards = 1/10 =
         0.1)
         # and store them in the counter "fractions".

         my_pets = ['cat', 'lizard', 'cat', 'dog', 'cat', 'snake', 'dog', 'cat', 'dog', 'parrot']

         #### YOUR CODE STARTS HERE ####

         pet_counter=Counter(my_pets)
         fractions={}
         for pet in pet_counter:
             fraction = float(pet_counter[pet]) /10
             fractions[pet] = fraction


         #### YOUR CODE ENDS HERE ####

         print("Testing: fraction of cats = %.1f" % fractions["cat"])
         print("CORRECT" if fractions["cat"] == .4 else "INCORRECT")
```

```
Testing: fraction of cats = 0.4
CORRECT
```

2.1 Applying Bayes rule: Which box did a ball come from?

In this exercise, we are interested in figuring out from which of the two boxes a ball of a certain color most likely came from. We are using Bayes rule to compute the probbaility of box A and box B given a ball of certain color and then we compare which one of these two probabilities is bigger.

Step 1: Inspect the data. How many different colors are there? How many balls of each color are in box A and in box B?

Hint: Turn the two lists `box_a` and `box_b` into Counters and print them.

```
In [14]: #### YOUR CODE STARTS HERE ####

         a_counter = Counter(box_a)
         b_counter = Counter(box_b)
         print(a_counter)
         print(b_counter)

         #### YOUR CODE ENDS HERE ####
```

```
Counter({'blue': 39, 'green': 27, 'orange': 23, 'red': 10, 'yellow': 1})
Counter({'red': 53, 'yellow': 25, 'green': 9, 'orange': 8, 'blue': 5})
```

Step 2: Compute the probability of each color in box A and each color in box B, i.e., compute $P(color \mid \text{box A})$ and $P(color \mid \text{box B})$ for each of the five colors. Store them in the counters `p_box_a` and `p_box_b`.

For example, you can compute the probability of picking a red ball from box 1 as:

$$P(red \mid \text{box A}) = \frac{\text{number of red balls in box A}}{\text{total number of balls in box A}}$$

```
In [15]: #### YOUR CODE STARTS HERE ####

         # Hint: This is very similar to Exercise 8.

         sum_a = sum(a_counter.values())
         sum_b = sum(b_counter.values())

         p_box_a = {}
         p_box_b = {}
         for color in a_counter:
             p_box_a[color] = float(a_counter[color]) / sum_a

         for color in b_counter:
             p_box_b[color] = float(b_counter[color]) / sum_b


         #### YOUR CODE ENDS HERE ####

         print(p_box_a)
         print(p_box_b)
```

```
{'orange': 0.23, 'green': 0.27, 'blue': 0.39, 'red': 0.1, 'yellow': 0.01}
{'red': 0.53, 'yellow': 0.25, 'green': 0.09, 'orange': 0.08, 'blue': 0.05}
```

Step 3: Now that we have the conditional probabilities for each color, we can apply Bayes rule to compute which box a ball of a certain color most likely came from. Fill in the blanks in this function.

As a reminder, the probability $P(box \mid color)$ is proportional to

$$P(box \mid color) \propto P(box) \times P(color \mid box)$$

```
In [16]: #### YOUR CODE STARTS HERE ####

         def likeliest_box(color):
```

```
        # The probability that someone picked a ball from box A or from box B
        # If we set both of these to 0.5, then this means that a box was chosen completely
    at random
        # Modify these values to see how the likelihood of the two boxes changes.
        prior_box_a = 0.5
        prior_box_b = 0.5

        # P(box A | color) is proportional to P(color | box A) * P(box A)
        # Hint use the prior_box_a variable and the p_box_a counter from the cell above.
        ratio = p_box_a[color] * prior_box_a / (p_box_b[color] * prior_box_b)

        # Which of the two boxes is likelier? Complete the following if statement
        # such that likely_box is assigned Box A if the ball came most likely from Box A
        # and Box B if it most likely came from the other box.
        if ratio > 1:
            likely_box = "Box A"
        else:
            likely_box = "Box B"

        return likely_box

    #### YOUR CODE ENDS HERE ####


    print("Balls in Box A:")
    print(Counter(box_a))
    print("Balls in Box B:")
    print(Counter(box_b))

    print("")

    colors = ["red", "green", "blue", "yellow", "orange"]

    for color in colors:
        print("A %s ball most likely came from %s" % (color, likeliest_box(color)))
```

```
Balls in Box A:
Counter({'blue': 39, 'green': 27, 'orange': 23, 'red': 10, 'yellow': 1})
Balls in Box B:
Counter({'red': 53, 'yellow': 25, 'green': 9, 'orange': 8, 'blue': 5})

A red ball most likely came from Box B
A green ball most likely came from Box A
A blue ball most likely came from Box A
A yellow ball most likely came from Box B
A orange ball most likely came from Box A
```

2.2 Which box did a sequence of balls most likely come from?

Now we are interested in a different question. We know that someone drew several balls from a single box, but we don't know from which one of the two. In other words, given a **list** of balls, we want to determine their likeliest origin.

The conditional probabilities remain the same, so all you have to do for this exercise is rewrite the *likeliest_box* function below.

As a reminder, $P(box \mid color_1, color_2, color_3, ...)$ is proportional to

$$P(box \mid color_1, color_2, color_3, ...) \propto P(box) \times P(box \mid color_1) \times P(box \mid color_2) \times P(box \mid color_3) \times ...$$

```
In [17]: #### YOUR CODE STARTS HERE ####
```

```
        def likeliest_box(colors):
            # The probability that someone picked a ball from box A or from box B
            # If we set both of these to 0.5, then this means that a box was chosen completely
        at random
            # Modify these values to see how the likelihood of the two boxes changes.
            prior_box_a = 0.5
            prior_box_b = 0.5

            # P(box A | color1, color2, color3, ...) is proportional to
            #      P(box A) * P(color1 | box A)  * P(color2 | box A) * P(color3 | box A) * ...
            # Hint use the prior_box_a variable and the p_box_a counter from above.

            p_box_a_colors = prior_box_a
            p_box_b_colors = prior_box_b

            for color in colors:
                p_box_a_colors *= p_box_a[color]
                p_box_b_colors *= p_box_b[color]

            ratio = p_box_a_colors / p_box_b_colors


            # Which of the two boxes is likelier? Complete the following if statement.
            if ratio > 1:
                likely_box = "Box A"
            else:
                likely_box = "Box B"

            return likely_box

        #### YOUR CODE ENDS HERE ####


        print("Balls in Box A:")
        print(Counter(box_a))
        print("Balls in Box B:")
        print(Counter(box_b))

        print("")

        sequences = [["red"], ["green"], ["blue"], ["yellow"], ["orange"],
                     ["red", "red", "green"],
                     ["red", "green", "green"],
                     ["blue", "red", "green", "yellow", "blue", "yellow", "yellow"],
                     ["yellow", "orange"],
                     ["yellow", "green"],
                     ["yellow", "green", "green", "green", "green"]]

        for seq in sequences:
            print("The sequence %s most likely came from %s" % (", ".join(seq),
        likeliest_box(seq)))
```

```
Balls in Box A:
Counter({'blue': 39, 'green': 27, 'orange': 23, 'red': 10, 'yellow': 1})
Balls in Box B:
Counter({'red': 53, 'yellow': 25, 'green': 9, 'orange': 8, 'blue': 5})

The sequence red most likely came from Box B
The sequence green most likely came from Box A
The sequence blue most likely came from Box A
The sequence yellow most likely came from Box B
The sequence orange most likely came from Box A
The sequence red, red, green most likely came from Box B
The sequence red, green, green most likely came from Box A
The sequence blue, red, green, yellow, blue, yellow, yellow most likely came from Box
```

```
B
The sequence yellow, orange most likely came from Box B
The sequence yellow, green most likely came from Box B
The sequence yellow, green, green, green, green most likely came from Box A
```

2.3 Optional Challenge: More boxes!

Up until now, we always assumed that there were just 2 boxes. But all of this can be extended to more boxes as well!

Re-implement the computation of probabilities for four boxes and implement a new *likeliest_box* method that can deal with more than two boxes.

This is a very challenging and open-ended problem. Think about how you could solve this and feel free to talk this through with us before you start implementing it.

Hint: One useful function for *likeliest_box* might be the *argmax* function (https://docs.scipy.org/doc/numpy/reference/generated/numpy.argmax.html).

```
In [18]: from numpy import argmax

         number_of_boxes = 4

         boxes = lib.get_box_contents(n_boxes=number_of_boxes)

         for i, box in enumerate(boxes):
             print("Box %d has %d balls." % (i + 1, len(box)))


         #### YOUR CODE STARTS HERE ####

         counter_1 = Counter(boxes[0])
         counter_2 = Counter(boxes[1])
         counter_3 = Counter(boxes[2])
         counter_4 = Counter(boxes[3])

         # Estimate the conditional probabilities for each box.

         p_box_1 = {}
         p_box_2 = {}
         p_box_3 = {}
         p_box_4 = {}

         for color in counter_1:
             p_box_1[color] = float(counter_1[color]) / len(boxes[0])
         for color in counter_2:
             p_box_2[color] = float(counter_2[color]) / len(boxes[1])
         for color in counter_3:
             p_box_3[color] = float(counter_3[color]) / len(boxes[2])
         for color in counter_4:
             p_box_4[color] = float(counter_4[color]) / len(boxes[3])

         #### YOUR CODE ENDS HERE ####


         def likeliest_box(colors):
             #### YOUR CODE STARTS HERE ####

             prior_box_1 = 0.25
             prior_box_2 = 0.25
             prior_box_3 = 0.25
             prior_box_4 = 0.25

             scores = [0] * 4
             scores[0] = prior_box_1
```

```
            scores[1] = prior_box_2
            scores[2] = prior_box_3
            scores[3] = prior_box_4

            for color in colors:
                scores[0] *= p_box_1[color]
                scores[1] *= p_box_2[color]
                scores[2] *= p_box_3[color]
                scores[3] *= p_box_4[color]

            print(scores)
            likeliest_box = argmax(scores)

            return "Box %d" % likeliest_box

            #### YOUR CODE ENDS HERE ####


        sequences = [["red"], ["green"], ["blue"], ["yellow"], ["orange"],
                     ["red", "red", "green"],
                     ["red", "green", "green"],
                     ["blue", "red", "green", "yellow", "blue", "yellow", "yellow"],
                     ["yellow", "orange"],
                     ["yellow", "green"],
                     ["yellow", "green", "green", "green", "green"]]

        for seq in sequences:
            print("The sequence %s most likely came from %s" % (", ".join(seq),
        likeliest_box(seq)))
```

```
Box 1 has 100 balls.
Box 2 has 100 balls.
Box 3 has 100 balls.
Box 4 has 100 balls.
[0.025, 0.1325, 0.0375, 0.0125]
The sequence red most likely came from Box 1
[0.0675, 0.0225, 0.0075, 0.0125]
The sequence green most likely came from Box 0
[0.0975, 0.0125, 0.0375, 0.0125]
The sequence blue most likely came from Box 0
[0.0025, 0.0625, 0.16, 0.0125]
The sequence yellow most likely came from Box 2
[0.0575, 0.02, 0.0075, 0.2]
The sequence orange most likely came from Box 3
[0.0006750000000000001, 0.00632025, 0.00016874999999999998, 3.125000000000001e-05]
The sequence red, red, green most likely came from Box 1
[0.0018225000000000003, 0.00107325, 3.3749999999999994e-05, 3.125000000000001e-05]
The sequence red, green, green most likely came from Box 0
[1.0266750000000003e-09, 4.658203125e-07, 6.635519999999999e-06,
1.953125000000001e-10]
The sequence blue, red, green, yellow, blue, yellow, yellow most likely came from Box
2
[0.000575, 0.005, 0.0048, 0.010000000000000002]
The sequence yellow, orange most likely came from Box 3
[0.000675, 0.005625, 0.0048, 0.0006750000000000001]
The sequence yellow, green most likely came from Box 1
[1.3286025000000003e-05, 4.100624999999999e-06, 1.2959999999999997e-07,
7.812500000000003e-08]
The sequence yellow, green, green, green, green most likely came from Box 0
```