C200 Programming Assignment №5

Professor M.M. Dalkilic
Computer Science
School of Informatics, Computing, and Engineering

Indiana University, Bloomington, IN, USA

March 3, 2023

Introduction

Due Date: 10:59 PM, Friday, March 10th 2023 Please note that both Autograder submission `https://c200.luddy.indiana.edu/` and pushing to GitHub **must be done before the deadline. We do not accept late submissions**. You are allowed to use Python's sum() function to sums lists or tuples.

Instructions

1. Make sure that you are **following the instructions** in the PDF, especially the format of output returned by the functions. For example, if a function is expected to return a numerical value, then make sure that a numerical value is returned (not a list or a dictionary). Similarly, if a list is expected to be returned then return a list (not a tuple, set or dictionary).

2. Test **debug** the code well (syntax, logical and implementation errors), before submitting to the Autograder. These errors can be easily fixed by running the code in VSC and watching for unexpected behavior such as, program failing with syntax error or not returning correct output.

3. Make sure that the **code does not have infinite loop (that never exits) or an endless recursion (that never completes)** before submitting to the Autograder. You can easily check for this by running in VSC and watching for program output, if it terminates timely or not.

4. Given that you already tried points 1-3, if you see that Autograder does not do anything (after you press 'submit') and waited for a while (30 seconds to 50 seconds), try refreshing the page or using a different browser.

5. Once you are done testing your code, comment out the tests i.e. the code under the __name__ == "__main__" section.

Problem 1: Entropy

In this problem, we will be calculating what's called *Entropy*. Entropy is used in ML, AI, vision, finance, *etc.* Entropy is a single number that describes how random the data is. If it's 0, then the data isn't random. If there are n objects and each one is equally probable, then the entropy is maximal: $\log(n)$. How might this be useful? Decisions are "better" the less entropy there is.

For this problem, we are only working with finite probabilities. We can think of probabilities as a list of numbers $p_0, p_1, \ldots, p_n$ such that

$$\begin{aligned} p_i &\geq 0, \quad i = 0, 1, \ldots, n && (1)\\ 1 &= p_0 + p_1 + p_2 + \cdots + p_n && (2)\\ &= \sum_{i=0}^{n} p_i && (3) \end{aligned}$$

We can make a list (we'll assume the items are of the same type and immutable) into a probability. For example, consider a list `x = ["a","b","a","c","c","a"]`.

1. gather the items uniquely (*hint: dictionary*)
2. count each time the item occurs
3. create a new list of the `count/len(dictionary)`

The list of probabilities would be y = [3/6,1/6,2/6] (a's probability is 3/6, b's probability is 1/6 and c's probability is 2/6). Observe that if you add up these numbers, they will sum to one.

We still need to show you *how* to calculate entropy:

$$\begin{aligned} entropy &= -(p_0 \log_2(p_0) + p_1 \log_2(p_1) + \cdots + p_n \log_2(p_n)) && (4)\\ entropy(y) &= -(\frac{3}{6}\log_2(3/6) + \frac{1}{6}\log(1/6) + \frac{2}{6}\log_2(2/6)) && (5)\\ &= -(.5(-1.0) + 0.17(-2.58) + .33(-1.58)) && (6)\\ &= 1.46 && (7) \end{aligned}$$

Because of continuity arguments we treat $\log_2(0) = 0$. Python's math module will correctly state this math error, so you'll **have to simply add 0 if the probability is 0**.

Deliverables Problem 1

- Take a non-empty list of objects and calculate the entropy.
- You **cannot** use the `count` function.
- Complete the functions for this problem.
- Hint: The makeProbability function is based on equation (3). Use the makeProbability function in your entropy() function.
- Round the output of entropy to two decimal places.

Problem 2: Run of 1s

For a financial instrument like stock, we want to buy and sell at the lowest and highest values, respectively. By observing trends, better decisions about buying and selling can be made. A "trend" is when when a sequence of values is **monotonic**. A sequence of values $v_0, v_1, v_2, \ldots, v_n$ is increasingly monotonic if

$$v_0 \leq v_1 \leq \cdots \leq v_n$$

The increasing monotonicity allows for selling at a higher price. Conversely, there is decreasing monotonicity. In this problem we'll be writing code to look for increasing monotonicity, but simplifying the problem by using only zero or one. In this problem, you'll take a list of 0s and 1s as an input. Your program will output the *longest* sequential run of 1s (no 0s encountered).

Output function.py

```
L([1, 0, 1, 0, 1, 0])
1

L([0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0])
2

L([1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1])
4
```

Deliverables Problem 2

- Complete the function.

Problem 3: Nine of Cattails

It's known that if you add the digits of any number, and that it eventually equals nine, that the original number is divisible by nine. Here is an example:

$$39789 \quad \rightarrow \quad 3+9+7+8+9=36 \rightarrow 3+6=9$$

It doesn't matter the order, *viz.,*

$$3+((9+7)+8)+9$$
$$3+((16)+8)+9$$
$$3+(7+8)+9$$
$$3+15+9$$
$$18+9$$
$$27 \rightarrow 9$$

We are left with only one digit which is 9 and hence the number 39789 is divisible by 9 (Infact, 9 * 4421 = 39789).

So we can see that the number 39789 is indeed divisible by 9 by following our method. Write a program that determines if a number is divisible by nine by using this method. You *cannot* simply divide by 9 and return True! You must continually add the numbers until a single digit is left; that digit will either be 9 or something else.

Caution: Remember that 'sum' is actually a built-in function in Python so we suggest that you store the results of addition in a variable named other than 'sum' to prevent errors or failures during testing.

sum = a + b + c (this is not a good practise because 'sum' is a reserved keyword in Python), rather
my_sum = a + b + c (this is safe to do)

Deliverables Problem 3

- Complete the functions **following** the algorithm described above.
- You can neither use the % sign in the program nor divide directly by 9.

Problem 4: Base 17

In this problem, you'll write a function that takes a number in base 17 (as a string) and return the decimal value. We'll simply extend base 16 by adding another digit G. Since F represents 15, G will represent 16. Interestingly, Python can interpret base 17 as we've imagined:

```
1 >>> int("G", 17)
2 16
3 >>> int("E2",17)
4 240
5 >>> int("10",17)
6 17
```

Programming Problem 4: Base 17

- Complete the function.
- You **cannot** use the integer function int(x,y) to convert to decimal. You must build the decimal value from powers of 17. You can still use int(x). Revising the lectures slides on conversion between number system can help to understand and solve this problem.

Problem 5: Coins

Sometimes problems are a lot simpler than you initially imagine. Write a function that takes an amount tot in dollars and returns a list:

$$[q, d, n, p]$$

where q is the number of quarters, d is the number of dimes, n is the number of nickels, and p is the number of pennies and

```
tot = q*.25 + d*.10 + n*.05 + p*.01
q+d+n+p is a minimum
```

For example, $2.24 = [8, 2, 0, 4] = [.25 \times 8 + .10 \times 2 + .05 \times 0 + .01 \times 4]$. Thus, your list should contain the fewest total coins possible that equals the dollar amount. You must only return non-negative integers!

Deliverables Programming Problem 5

- Complete the function
- You must return non-negative integers.

Problem 6: Models: Least Squares

A model is a characterization of something. A mathematical model is a formula that describes a relationship among values. The simplest mathematical formula is a linear equation in 2D:

$$0 \quad = \quad Ax + By + C$$

We usually rewrite the equation above as:

$$y \quad = \quad mx + b$$

This means we can determine any value of y by using x. The relationship is **linear** because we're only using $mx + b$, not for example, x^2. Least squares, as we've discussed in lecture is several hundred years old, but remains the most popular modeling for simple relationships. We are given a set of pairs:

$$data \quad = \quad \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$$

and want to build the "best" function $f(x) = \hat{m}x + \hat{b}$. where the $\hat{}$ (called hat) means we're approximating a value (we don't know the actual m, b. Without going through the math, there is now a simple procotol to build $\hat{m}, \hat{b}$:

$$\begin{aligned}
data &= \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\} \\
xy_p &= \sum_{i=0}^{n} x_i y_i \\
x_s &= \sum_{i=0}^{n} x_i \\
y_s &= \sum_{i=0}^{n} y_i \\
x_{sq} &= \sum_{i=0}^{n} x_i^2 \\
\\
S_{xy} &= xy_p - \frac{x_s y_s}{n} \\
S_{xx} &= x_{sq} - \frac{x_s^2}{n} \\
\hat{m} &= \frac{S_{xy}}{S_{xx}} \\
\hat{b} &= \frac{y_s - \hat{m}x_s}{n} \\
f(x) &= \hat{m}x + \hat{b}
\end{aligned}$$

While this seems like a lot of loops, using Python comprehension and lambda, we can make quick work of this. Observe each summation is the same–the only thing changing is f. We could write a function that takes the data and function returning the value–here are a couple for y_s and x_{sq}.

```
...
def r(data,f):
        return sum([f(x,y) for x,y in data])
...
y_s = r(data,lambda x,y:y)
x_sq = r(data,lambda x,y:x**2)
...
```

Here's a run on some data:

```
    data6 = [(2,1),(5,2),(7,3),(8,3)]

    m_hat, b_hat = std_linear_regression(data6)
    print(m_hat,b_hat)
    plt.plot([x for x,_ in data6],[y for _,y in data6],'ro')
    plt.plot([x for x,_ in data6],[m_hat*x + b_hat for x,_ in data6],'b')
    plt.title(r"Least Squares: $\hat{m}=0.36,\hat{b}=0.27$")
    plt.ylabel("Y")
    plt.xlabel("X")
    plt.show()
```

with output:

```
0.36 0.27
```

and graphic:

Note: For this problem, we have already given the code under (__name__ == __main__) to plot the graph as shown above. After completing this problem, you can un-comment the code to draw the graph. Remember to comment it back before submitting to the Autograder. The Autograder can not draw graphical plots in the web bowser, and thereofre your submission attempt may fail with errors even though the code is correct, so please comment the code related to the plot before making final submission.

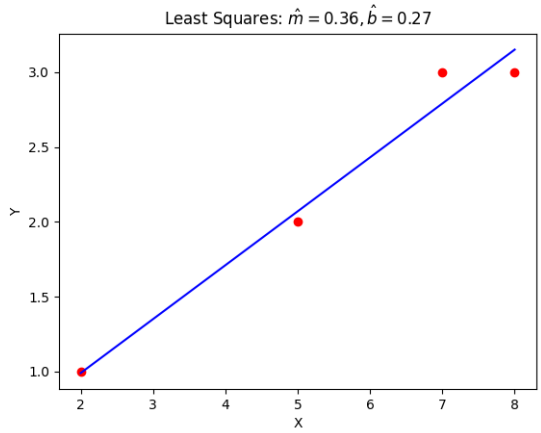

Figure 1: Least squares function (blue) and data (red). The least-squares regression models the linear relationship between the data points.

Deliverables Programming Problem 6

- Complete the function–you are encouraged to use comprehension and lambda, but are free to implement this as you wish
- You cannot use any existing tools for regression–we'll be using those later
- Round the final values of $\hat{m}$ and $\hat{b}$ to two decimal places

Paired Programming Partners

aaberma@iu.edu, mostrodt@iu.edu, imaychru@iu.edu
actonm@iu.edu, pvinod@iu.edu
adagar@iu.edu, btpfeil@iu.edu
brakin@iu.edu, chnbalta@iu.edu
sakinolu@iu.edu, jonhick@iu.edu
moalnass@iu.edu, tflaa@iu.edu
anderblm@iu.edu, isarmoss@iu.edu
jaybaity@iu.edu, ajneel@iu.edu
nbakken@iu.edu, megofolz@iu.edu
nokebark@iu.edu, anthoang@iu.edu
nwbarret@iu.edu, cartwolf@iu.edu
sbehman@iu.edu, jvalleci@iu.edu
jadbenav@iu.edu, alescarb@iu.edu
arjbhar@iu.edu, aibitner@iu.edu
dombish@iu.edu, cjromine@iu.edu
abolad@iu.edu, crfmarti@iu.edu
jacbooth@iu.edu, davthorn@iu.edu
joshbrin@iu.edu, sabola@iu.edu
neybrito@iu.edu, trewoo@iu.edu
nbulgare@iu.edu, roelreye@iu.edu
ivycai@iu.edu, jl335@iu.edu
bcarl@iu.edu, svaidhy@iu.edu
ecastano@iu.edu, rmatejcz@iu.edu
cheng47@iu.edu, dwinger@iu.edu
kbchiu@iu.edu, danwils@iu.edu
hc51@iu.edu, dy11@iu.edu
joracobb@iu.edu, hnichin@iu.edu
sgcolett@iu.edu, jacosby@iu.edu
swcolson@iu.edu, omilden@iu.edu
joecool@iu.edu, lenonti@iu.edu
quecox@iu.edu, sjxavier@iu.edu
jacuau@iu.edu, ajarillo@iu.edu
aadidogr@iu.edu, bpoddut@iu.edu
siqidong@iu.edu, alexschu@iu.edu
wdoub@iu.edu, ss126@iu.edu
dud@iu.edu, marystre@iu.edu
marbelli@iu.edu, bensokol@iu.edu
moesan@iu.edu, gsprinkl@iu.edu
abdufall@iu.edu, jl263@iu.edu

stfashir@iu.edu, nmonberg@iu.edu
sfawaz@iu.edu, tyfeldm@iu.edu
jfearri@iu.edu, ekkumar@iu.edu
lgflynn@iu.edu, hvelidi@iu.edu
bgabbert@iu.edu, chrinayl@iu.edu
mgambett@iu.edu, mgorals@iu.edu
mdgamble@iu.edu, jcpilche@iu.edu
ogift@iu.edu, swa5@iu.edu
jegillar@iu.edu, parisbel@iu.edu
egillig@iu.edu, yijwei@iu.edu
agoldswo@iu.edu, sousingh@iu.edu
davgourl@iu.edu, bashih@iu.edu
qugriff@iu.edu, peschulz@iu.edu
aguarda@iu.edu, achimeba@iu.edu
sihamza@iu.edu, criekki@iu.edu
lsherbst@iu.edu, nkmeade@iu.edu
qhodgman@iu.edu, almoelle@iu.edu
hollidaa@iu.edu, dpepping@iu.edu
tifhuang@iu.edu, sowmo@iu.edu
rythudso@iu.edu, nireilly@iu.edu
bradhutc@iu.edu, yz145@iu.edu
ghyatt@iu.edu, jmom@iu.edu
jackssar@iu.edu, parkecar@iu.edu
sijatto@iu.edu, hmerrit@iu.edu
ljianghe@iu.edu, fimitch@iu.edu
coldjone@iu.edu, jmissey@iu.edu
vilokale@iu.edu, aketcha@iu.edu
sjkallub@iu.edu, saseiber@iu.edu
dilkang@iu.edu, sjvaleo@iu.edu
pkasarla@iu.edu, greymonr@iu.edu
katzjor@iu.edu, georliu@iu.edu
dk80@iu.edu, jorzhang@iu.edu
eddykim@iu.edu, mszczas@iu.edu
arykota@iu.edu, cmulgrew@iu.edu
delkumar@iu.edu, aalesh@iu.edu
akundur@iu.edu, kvanever@iu.edu
aladkhan@iu.edu, elyperry@iu.edu
jolawre@iu.edu, esmmcder@iu.edu
sl92@iu.edu, rpmahesh@iu.edu
jacklian@iu.edu, caitreye@iu.edu
klongfie@iu.edu, leplata@iu.edu

dloutfi@iu.edu, grschenc@iu.edu
jasoluca@iu.edu, zisun@iu.edu
imalhan@iu.edu, jyamarti@iu.edu
lmamidip@iu.edu, masmuell@iu.edu
cmarcucc@iu.edu, avpeda@iu.edu
kalomart@iu.edu, patel88@iu.edu
jarymeln@iu.edu, macsvobo@iu.edu
cartmull@iu.edu, zhaozhe@iu.edu
ornash@iu.edu, schwani@iu.edu
chnico@iu.edu, snuthala@iu.edu
arnpate@iu.edu, webejack@iu.edu
yasmpate@iu.edu, edshipp@iu.edu
rair@iu.edu, voram@iu.edu
psaggar@iu.edu, shethsu@iu.edu
shahneh@iu.edu, jonvance@iu.edu
tshore@iu.edu, johwarre@iu.edu
vthakka@iu.edu, dannwint@iu.edu
mw154@iu.edu, jwescott@iu.edu
aw149@iu.edu, wuyul@iu.edu