Coding Problem Set 1 - Probability Review

Joshua Mangelson

September 8, 2021

Purpose: The purpose of this lab is to give you hands on experience implementing, and the ability to play around with, several fundamental probability concepts. We will be using these concepts frequently throughout the course, so it is essential that you gain a good understanding of them now.

This assignment is broken into three main sections.

0 Setup and Initial Instructions

0.1 Pulling the Code

You will be able to access the code for each of the labs in this course via Git. For this lab specifically, you will be able to access it via the following link: `https://bitbucket.org/byu_mobile_robotic_systems/lab1-prob-review/src/master/`

0.2 Setting Up Your Environment

The programming language for this lab is Python. We assume you will be using Python3. The code has the following dependencies:

1. matplotlib - (this is a fundamental python graphing library)
2. pytest - (several tests are provided that will allow you to verify the operation of your code)

Conda is a very useful python environment manager, and can be used on Windows, Linux, and Mac OS systems. If you will be using conda you can setup your environment as follows (for this example the environment has been given the name of 'pr', you can name your environment whatever you would like, simply swap out 'pr' wherever it appears with whatever name you prefer):

```
conda create -n pr python=3.8 -y
conda activate pr
pip install matplotlib pytest
```

Alternatively if you are not using conda, on Ubuntu, you should be able to install what you need via the following commands:

```
sudo apt-get install python3
sudo apt-get install python3-pip
pip3 install matplotlib
```

For other operating systems, instructions for setup without the use of conda are not provided here. Please just ensure you have access to Python3 and have the necessary dependencies installed.

Important Note: Throughout the remainder of this document there are instances where python code is called as follows:

```
python3 [python file path and name].py
```

If you are using conda, instead of leading with 'python3' you should lead with 'python', after making sure that you have the appropriate environment active (by default conda starts up with the 'base' environment active, you can switch to other environments with the 'conda activate [environment name]' command).

0.3 Code Overview

The code needed for this lab is found inside the **probability_review/** folder. Within this folder are 5 main files with the **.py** extension.

- **__init__.py** - this file defines the package and can be ignored for the sake of this assignment.
- **discrete_probability_core.py** - this file defines some basic classes including:
  - **DiscreteExperiment** - this class is a base class for a discrete experiment. Example experiments will extend this class and define the **enumerate_outcomes** function to enumerate the set of possible outcomes for an experiment along with the probability that they will occur.
  - **ProbabilityMassFunction** - this class defines an object that represents the PMF of a discrete random variable.
  - **JointProbabilityMassFunction** - this class defines an object that represents the Joint PMF of two discrete random variables.
- **part1.py** - this file defines two classes and several functions:
  - **DiceRoll** and **TwoDiceRoll** - these classes are example random experiments and enumerate the outcomes that occur from rolling either one or two dice and their associated probabilities
  - **two_dice_sum**, **two_dice_difference**, **doubles_rolled**, **two_dice_sum_plus_one**, **weighted_two_dice_sum**, and **two_dice_true_difference** - each of these functions define an example of a random variable (or informally a function mapping an outcome of an experiment to a real number)
  - **evaluate_pmf**, **expected_value**, and **variance** - each of these functions are incomplete and will need to be implemented by you in Part 1 of this assignment.
  - **main** - lastly, this function is the function that is run if you type the following command in a terminal:

```
python3 probability_review/part1.py
```

- **part2.py** - This file contains 4 functions:
  - **evaluate_joint_pmf**, **expected_value**, and **covariance** - each of these functions are incomplete and will need to be implemented by you in Part 2 of this assignment
  - **main** - again, this function is the function that is run if you type the following command in a terminal:

```
python3 probability_review/part2.py
```

- **part3.py** - This file contains 3 functions:
  - **marginalize_out** and **conditionalize_against** - each of these functions are also incomplete and will need to be implemented by you in Part 3 of this assignment
  - **main** - again, this function is the function that is run if you type the following command in a terminal:

```
python3 probability_review/part3.py
```

0.4 Instructions

The following sections will describe what needs to be completed for this lab.

The system is setup such that the dice rolls can be n-sided dice (3 sides is the minimum that is accepted). For the writeup make sure that you are using six-sided dice (this is the default, if you do not change anything it will always run with six-sided dice). Some simple pytest scripts are included for each part. Those tests will use four-sided dice, once again you do not need to change anything, just note that the printed output of the main functions in each part will not match with what the tests are comparing against. However, feel free to experiment with any other number of sided dice along the way, to either help you better understand the functions, or to debug any issues that you occur with simpler examples. Just make sure that you return back to six-sided dice for the writeup.

The " **TODO:** " indicator denotes something that you need to do. The " **QUESTION:** " indicator denotes a question that should be answered inside your writeup.

1 PMFs, Expected Value, and Variance

Random variables are functions that map a set of possible outcomes of an experiment to a real number. In this lab we will use the roll of two (six-sided) dice as an experiment. Various random variables are defined on lines 91 to 116 in **part1.py**.

1.1 Probability Mass Functions

A probability distribution is a function that maps possible values a random variable can take to the probability they are to occur. For discrete random variables this function is called a probability mass function and the sum of the probabilities for all outcomes must be equal to 1. For continuous random variables, this function is called a probability density function and the integral over the range of possible values must be equal to 1.

For this lab, we will focus on discrete random variables.

TODO: Implement the **evaluate_pdf** method in **part1.py**, ensuring that any exceptions described in the function description are raised (see **discrete_probability_core.py** for examples on raising exceptions).

1.2 Expected Value

The moments of a random variable are statistics that give us information about a probability distribution.

The expected value or mean is one of the primary statistics used to describe a random variable.

TODO: Implement the **expected_value** function to evaluate the expected value of a ramdom variable from its PMF, ensuring that any exceptions described in the function description are raised.

1.3 Variance

The variance is a statistic that describes how much a random variable "varies" or deviates from the mean.

TODO: Implement the **variance** function to evaluate the variance of a random variable from its PMF, ensuring that any exceptions described in the function description are raised.

1.4 Wrap Up

TODO: Run the **part1.py** file to see the output of your implemented functions for the sum of two six-sided dice, the difference of two dice, and the indicator random variable for rolling doubles. Include the output in your writeup.

TODO: Run the tests for part 1 ('pytest test/test_part1.py' with '-v' being an optional addition that will make the output more verbose) and correct any problems. All tests should pass if your functions are complete and correct. Include the output of '`pytest -v -rN --tb=no --no-header test/test_part1.py`' in your writeup.

QUESTION: Why does the PMF for the sum of two dice look the way it does? Can you explain why a value of 7 is more likely than a value of 10?

2 Jointly Distributed Random Variables and Covariance

2.1 Jointly Distributed Random Variables

A joint distribution of random variables models the relationship between multiple random variables. Specifically it describes the probability that different random variables take on specific values at the same time.

TODO: Implement the **evaluate_joint_pmf** function in **part2.py**, ensuring that any exceptions described in the function description are raised.

2.2 Moments of Jointly Distributed Random Variables

Just like individual random variables, joint PMFs also have moments such as expectation and covariance. For joint random variables, these are often represented using vectors or matrices as opposed to individual random variables.

2.2.1 Expectation

TODO: Implement the **expected_value** function in **part2.py** to evaluate the expected value of a pair of jointly distributed random variables from the joint PMF, ensuring that any exceptions described in the function description are raised.

2.2.2 Covariance

The covariance matrix is the jointly distributed random variable equivalent of the variance for a single random variable. The elements of the diagonal of the covariance matrix are the variances of the individual random variables. The off diagonal elements are the covariances and they how "correlated" the different variables are. For example, a high positive covariance value means that if one variable goes up the other will likely as well. A covariance of 0 means the variables are uncorrelated meaning that there is not necessarily a pattern between whether or not one variable will increase when another variable does.

TODO: Implement the **covariance** function in **part2.py** to evaluate the covariance matrix of a pair of jointly distributed random variables from the joint PMF, ensuring that any exceptions described in the function description are raised.

2.2.3 Wrap Up

TODO: Run the **part2.py** file to see the output of your implemented functions for several pairs of random variables. Include the output in your writeup.

TODO: Run the tests for part 2 ('pytest test/test_part2.py') and correct any problems. All tests should pass if your functions are complete and correct.
Include the output of '`pytest -v -rN --tb=no --no-header test/test_part2.py`' in your writeup.

QUESTION: For each pair of jointly distributed random variables, are the two variables correlated with one another? How do you know? Does this make sense?

3 Marginalization and Conditioning

The nice thing about joint distributions is that it enables us to reason about individual random variables as well as sets of random variables. This is mainly often done via marginalization and conditioning.

3.1 Marginalization

Marginalization is how you remove random variables you no longer care about from the joint distribution.

TODO: Implement **marginalize_out** in **part3.py**, ensuring that any exceptions described in the function description are raised.

3.2 Conditioning

Conditioning enables us to evaluate the conditional distribution describing the value of one random variable given another.

TODO: Implement **condition_against** in **part3.py**, ensuring that any exceptions described in the function description are raised.

3.2.1 Wrap Up

TODO: Run the **part3.py** file to see the output of your implemented functions for the joint PMF of the sum and difference of a roll of two die. Include this output in your writeup.

TODO: Run the tests for part 3 ('pytest test/test_part3.py') and correct any problems. All tests should pass if your functions are complete and correct.
Include the output of '`pytest -v -rN --tb=no --no-header test/test_part3.py`' in your writeup.

QUESTION: Do the marginal PMFs match the PMFs you generated in part 1?

QUESTION: Please describe the output of the conditional PMFs. Why does the output make sense?

TODO: Upload your writeup and code to Learning Suite. **The grading for your lab work will be performed automatically with another program. In order to ensure that you receive the grade that you deserve, please be sure to read through the following instructions on lab submission carefully!**

4 How to Submit your Lab Work

Send both your writeup and the 'probability_review' folder containing your code to a compressed (.zip) folder.

So the final file structure of your submission should be:

```
submission.zip
├── writeup.pdf
└── probability_review
    ├── __init__.py
    ├── discrete_probability_core.py
    ├── part1.py
    ├── part2.py
    └── part3.py
```

The name of the .zip folder does not matter, as well as the name of your writeup (though your writeup should be a .pdf file).

However, the folder name of 'probability_review' and the filenames of the files therein are very important! So if for some reason you decided to rename any of those items while you were working on the lab, please ensure that they are changed back to their original names before completing your submission.